Google Cloud

ST

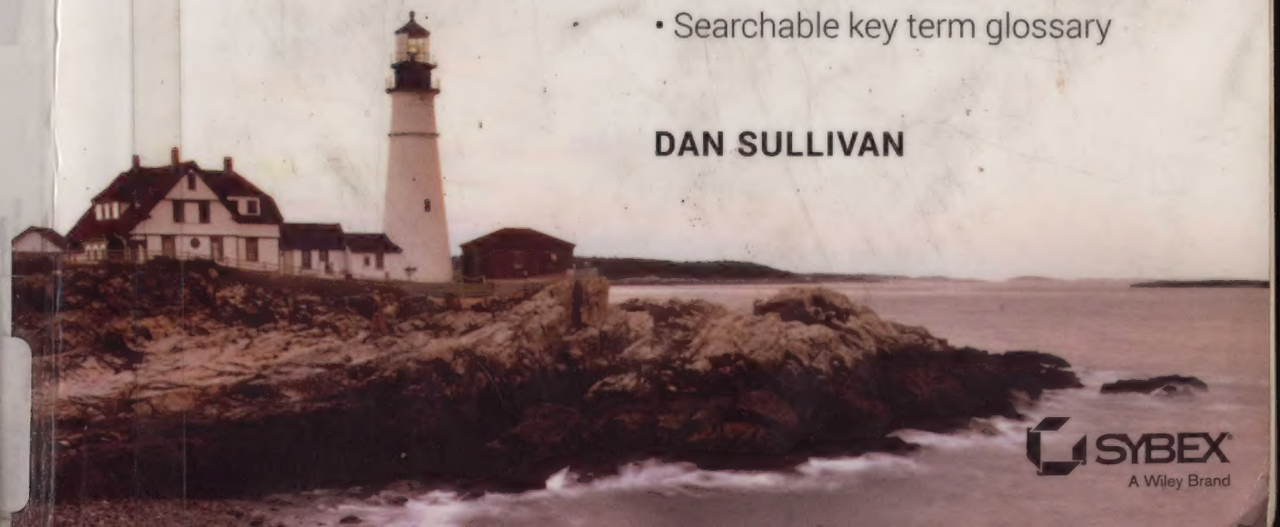
Official Google Cloud Certified

Associate Cloud Engineer Study Guide

Includes interactive online learning environment and study tools:

- 2 custom practice exams
- More than 100 electronic flashcards
- Searchable key term glossary

DAN SULLIVAN



SYBEX
A Wiley Brand

Docker: Up & Running

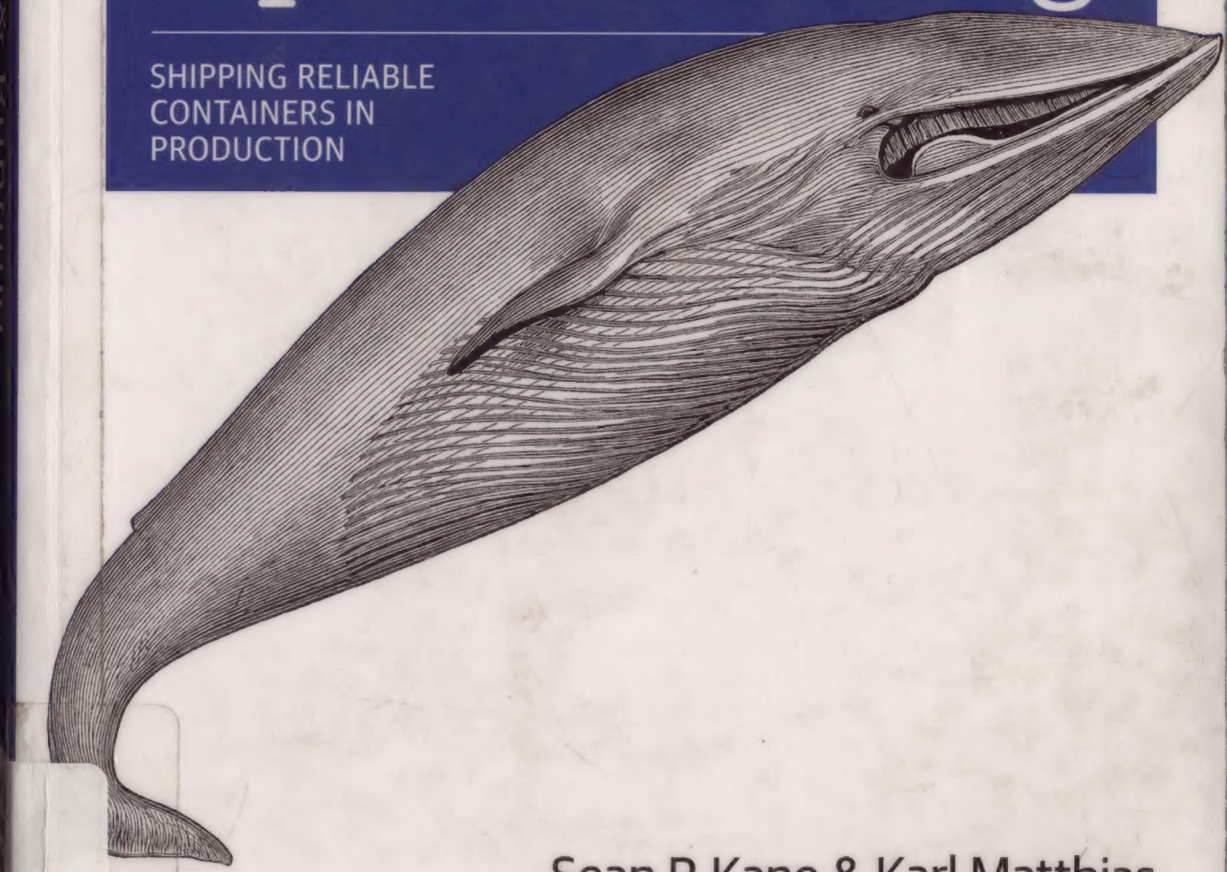
O'REILLY®

2nd Edition

ST

Docker Up & Running

SHIPPING RELIABLE CONTAINERS IN PRODUCTION



Sean P. Kane & Karl Matthias

The Official Study Guide to Prepare You for the Google Associate Cloud Engineer Exam—and More

Google Cloud Platform is a leading public cloud used by businesses, organizations, and individuals. Certified Associate Cloud Engineers have demonstrated the knowledge and skills needed to deploy and operate infrastructure, services, and networks in the Google cloud. With coverage of all of the exam objectives, you'll prepare for the exam smarter and faster with Sybex, but more than that—it helps you understand how to meet the day-to-day challenges faced by cloud engineers on the job. With Exam Essentials in every chapter and exercises to help you reinforce your knowledge, plus the exclusive Sybex online learning environment and test bank, it prepares you for both certification and career confidence.

Coverage of 100% of all exam objectives in this *Study Guide* means you'll be ready for:

- Selecting the right Google service based on the application
- Computing with Cloud VMs
- Computing with Kubernetes
- Managing Kubernetes and VMs
- Computing with and managing App Engines
- Planning and deploying Storage
- Networking
- Configuring Access and Security

DAN SULLIVAN is a software architect specializing in big data, machine learning, and cloud computing. He is currently working on real-time analysis of application and system monitoring data. Dan is a Google Cloud Certified Associate Cloud Engineer and has extensive experience with Google Cloud and other cloud platforms. Dan is the author of three books and numerous articles. He is an instructor with LinkedIn Learning where he develops courses on data science, machine learning, and data management.

www.sybex.com
www.wiley.com/go/sybextestprep

Cover Design: Wiley
Cover Image: © Getty Images Inc./Jeremy Woodhouse

SYBEX
A Wiley Brand

Google Cloud

$50.00 USA
$60.00 CAN

Also available as an e-book

Take your exam prep to the next level with Sybex's superior interactive online study tools. To access our learning environment, simply visit **http://www.wiley.com/go/sybextestprep**, register your book to receive your unique PIN, and instantly gain a year of FREE access to:

- **Interactive test bank with 2 practice exams.** Practice exams help you identify areas where further review is needed. 100 questions total!
- **100+ electronic flashcards** to reinforce learning and last-minute prep before the exam
- **Comprehensive glossary in PDF format** gives you instant access to the key terms so you are fully prepared

CATEGORY
COMPUTERS/Certification Guides

ISBN 978-1-119-56441-6
55000
9 781119 564416

O'REILLY®

Docker: Up & Running

Docker is rapidly changing the way organizations deploy software at scale. However, understanding how Linux containers fit into your workflow and getting the integration details right are not trivial tasks. With the updated edition of this practical guide, you'll learn how to use Docker to package your applications with all of their dependencies and then test, ship, scale, and support your containers in production.

This edition includes significant updates to the examples and explanations that reflect the substantial changes that have occurred over the past couple of years. Sean Kane and Karl Matthias have added a complete chapter on Docker Compose, deeper coverage of Docker Swarm mode, introductions to both Kubernetes and AWS Fargate, examples on how to optimize your Docker images, and much more.

- Learn how Docker simplifies dependency management and deployment workflow for your applications
- Start working with Docker images, containers, and command-line tools
- Use practical techniques to deploy and test Docker containers in production
- Debug containers by understanding their composition and internal processes
- Deploy production containers at scale inside your data center or cloud environment
- Explore advanced Docker topics, including deployment tools, networking, orchestration, security, and configuration

Sean P. Kane is a lead site reliability engineer at New Relic. He has had a long career in production operations, serving in many diverse roles across a broad range of industries.

Karl Matthias is Director of Cloud and Platform Services at InVision. He has worked as a developer, distributed systems architect, systems administrator, and network engineer at everything from startups to Fortune 500 companies.

"*Docker: Up and Running* moves past the Docker honeymoon and prepares you for the realities of running containers in production."

—Kelsey Hightower
Staff Developer Advocate, Google Cloud Platform

"We're in the midst of an explosion in the use of container technology. *Docker: Up and Running* takes you from the basic underlying concepts to invaluable practical lessons learned from running Docker at scale."

—Liz Rice
Chief Technology Evangelist, Aqua Security

"This is one of my favorite books on Docker! I always recommend it to anyone who is looking at Docker for the first time."

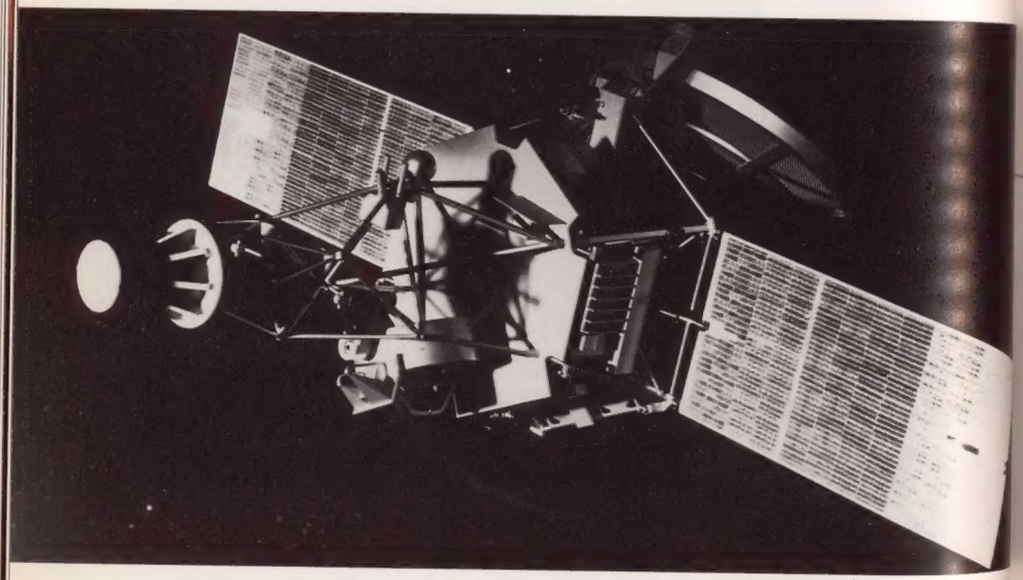
—Ksenia Burlachenko
Senior Backend Engineer

US $49.99 CAN $65.99
ISBN: 978-1-492-03673-9
9 781492 036739 54999

Twitter: @oreillymedia
facebook.com/oreilly

plore space. In just a few years, the first Russian cosmonauts orbited the Earth, and they were followed into space by the first American astronauts.

The early accomplishments of the U.S. manned space program gained the world's attention and overshadowed an equally important event. In 1962, less than five years after the first satellite launch, the first successful interplanetary spacecraft, *Mariner 2,* flew past Venus. The spacecraft traveled millions of miles through space and passed by Venus at a distance of 21,000 miles from its cloudtops. *Mariner 2*'s instruments gave astronomers their first close-up indications of the kind of world Venus really is.



An early *Mariner* spacecraft.

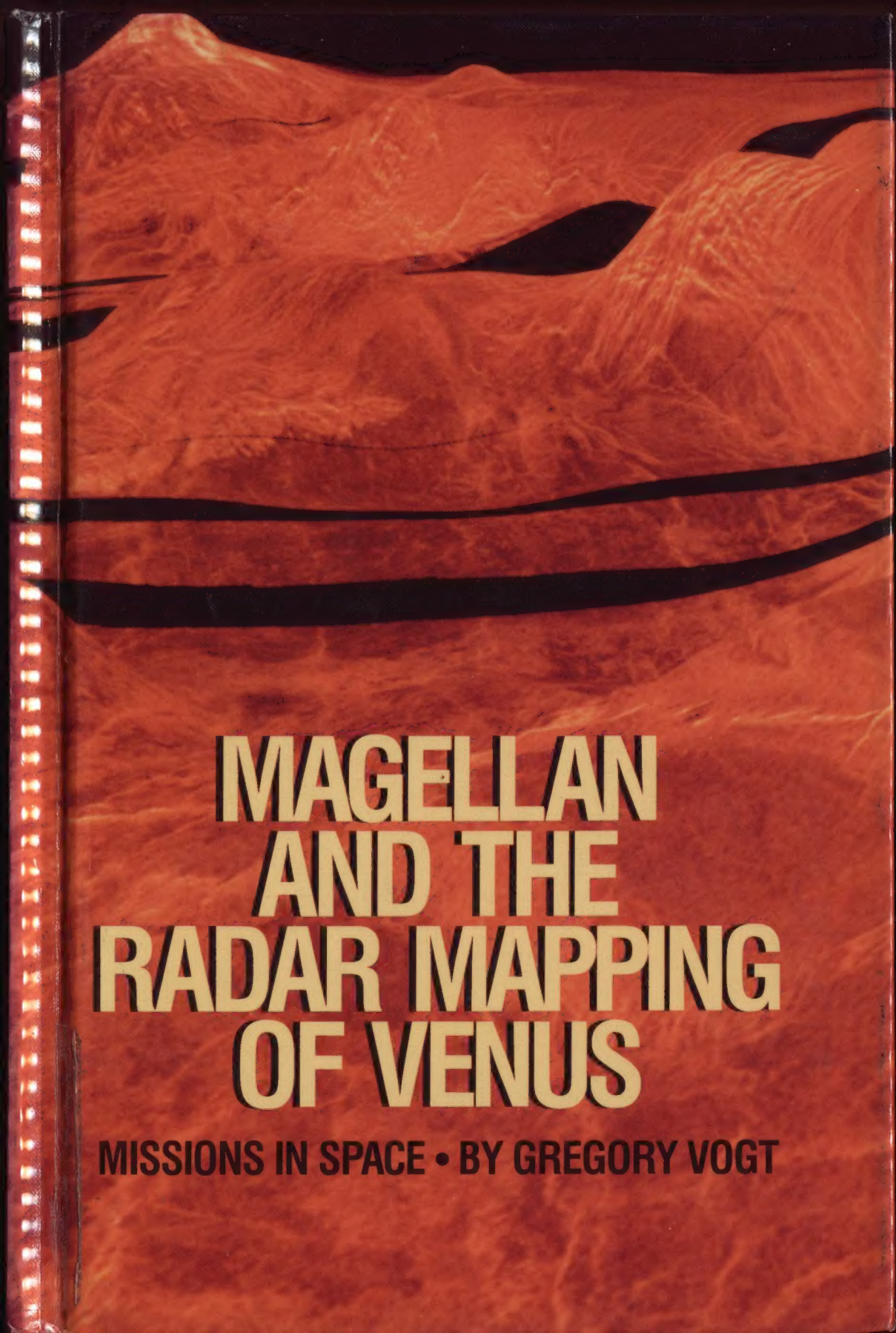
Mariner 2's flight past Venus was the first of a long string of space voyages to study the veiled planet. Some spacecraft, such as *Mariner 2,* had only a short encounter with Venus. Later, landers were sent to the planet's surface, balloons were dropped into its atmosphere, and other spacecraft went into orbit around it. In all, 20 spacecraft were sent to Venus between 1962 and 1986.

In 1990, a new spacecraft arrived at Venus, and it maneuvered itself into an elliptical (egg-shaped) orbit. This spacecraft, called *Magellan,* was sent by the U.S. National Aeronautics and Space Administration (NASA) to continue our exploration of Venus and learn what the planet's surface is really like.

Magellan arrived on August 10, 1990, and began its long-term studies of Venus. A small braking rocket on the spacecraft slowed it in its interplanetary course so that Venus's gravity could capture it and pull it into orbit. After its capture, *Magellan* used powerful radar waves to penetrate the clouds and map much of the planet's surface.

NASA sent *Magellan* to Venus because astronomers wanted to learn more about the planet. Solving mysteries is what astronomers try to do. Each discovery they make helps them to understand the story of how our Solar System came into being and, more important, how life came to be.

There is a more practical side to the scientists' interest in Venus. Because it is the nearest and perhaps the most similar planet to Earth, under-



MAGELLAN AND THE RADAR MAPPING OF VENUS

MISSIONS IN SPACE • BY GREGORY VOGT

[*Part I*]

What Are Social Patterns?

In the past 20 years, we have watched the proliferation of Internet technology spread across the globe and have been immersed in the creation of tools and interactive experiences to help people navigate their way to information, find other people, and create their own places in the digital space. As both designers and participants, we have seen the rise and fall of the first wave (the dotcom boom and bust), experienced firsthand the explosion of Web 2.0 and social media, and witnessed the coming age of the Internet of Things (IoT).

These electronic connections and social tools are changing the way we interact with one another and our environments. We believe that these tools can be designed and simplified to help normal people expand their digital experiences with others. These social patterns of behavior and the interfaces to support them have emerged and continue to evolve as we find better ways to bring people together.

Social patterns are the components and pieces of interactivity that are the building blocks of social experiences. They are the best practices and principles we have seen emerge from hundreds of sites and applications with social features or focus. They are the emergent interaction patterns that have become the standard way for users to interact with their content and with the people who matter most to them.

[1]

Mommy, What's a Social User Experience Pattern?

I have a dream for the Web... and it has two parts. In the first part, the Web becomes a much more powerful means for collaboration between people. I have always imagined the information space as something to which everyone has immediate and intuitive access, and not just to browse, but to create. Furthermore, the dream of people-to-people communication through shared knowledge must be possible for groups of all sizes, interacting electronically with as much ease as they do now in person.

TIM BERNERS-LEE, *WEAVING THE WEB* **(1999)**

A Little Social Backstory...

SOCIAL DESIGN FOR INTERACTIVE digital spaces has been around since the earliest bulletin-board systems. The most famous being The Well (1985), which was described by *Wired* magazine in 1997* as "the world's most influential online community" and predated the World Wide Web and browser interfaces by several years.

* Hafner, Katie. 1997. "The Epic Saga of The Well: The World's Most Influential Online Community (And It's Not AOL)." *Wired*, May 5, 1997. http://www.wired.com/wired/archive/5.05/ff_well_pr.html.

The Well

The Well began in 1985 on a Digital Equipment Corporation VAX system and a series of modems. Conceived by Stewart Brand and Larry Brilliant, Brand had a simple idea: "take a group of interesting people, give them the means to stay in continuous communication with one another, stand back, and see what happens." He also had the idea that a community created online through written dialogue could be strengthened through offline, face-to-face meetings, and he set out to combine the two quite successfully.

From the 1997 *Wired* article: "But probably the most important of Brand's early convictions for The Well was that people should take responsibility for what they said. There would be no anonymity; everyone's real name would be available on the system, linked to his or her login. Brand came up with a credo that would, through the years, spark no end of debate: 'You own your own words.' That proviso greeted members each time they logged on. 'I was doing the usual, considering what could go wrong,' he recalls. 'One thing would be people blaming us for what people said on The Well. And the way I figured you get around that was to put the responsibility on the individual.'"

Since the beginning of connected computers, we have tried to have computer-mediated experiences between people. As Clay Shirky notes in a 2004 *Salon* article, "Online social networks go all the way back to the Plato BBS 40 years ago!"

In the early days of the Web, social experiences were simply called *community* and generally consisted of message boards, groups, list-servs, and virtual worlds. Amy Jo Kim, author and community expert, calls these "place-centric" gathering places. Community features allowed users to talk and interact with one another, and the connection among people was usually based on the topic of interest that drew them to the site in the first place. Communities formed around interests, and relationships evolved over time. There was little distinction between the building of the tools to make these gatherings possible and the groups of people who made up the community itself. Bonds were formed in this space but generally didn't exist in the real (offline) world.

PLATO*

"PLATO (Programmed Logic for Automated Teaching Operations) originated in the early 1960s at the Urbana campus of the University of Illinois. Professor Don Bitzer became interested in using computers for teaching, and together with some colleagues, he founded the Computer-based Education Research Laboratory (CERL).

"The sense of an online community began to emerge on PLATO in 1973–74, as Notes, Talkomatic, 'term-talk', and Personal Notes were introduced in quick succession. People met and got acquainted in Talkomatic, and carried on romances via 'term-talk' and Personal Notes. The release of Group Notes in 1976 gave the community fertile new ground for growth, but by that time it was already well established. The community had been building its own additions to the software infrastructure in the form of multiplayer games and alternative online communications. One such program was Pad, an online bulletin board where people could post graffiti or random musings. Another was Newsreport, a lighthearted online newspaper published periodically by Bruce Parello, aka The Red Sweater."

* Excerpted from David R. Woolley's 1994 article "PLATO: The Emergence of Online Community" (*http://thinkofit.com/plato/dwplato.htm#community*). An earlier version of this article appeared in the January 1994 issue of *Matrix News*.

The interfaces and interaction design for these types of tools were all over the board—from graphical representations such as eWorld (see Figure 1-1) to scary-looking, only-for-early-adopters, text-only BBSs, to the simple forms of AOL chat rooms.

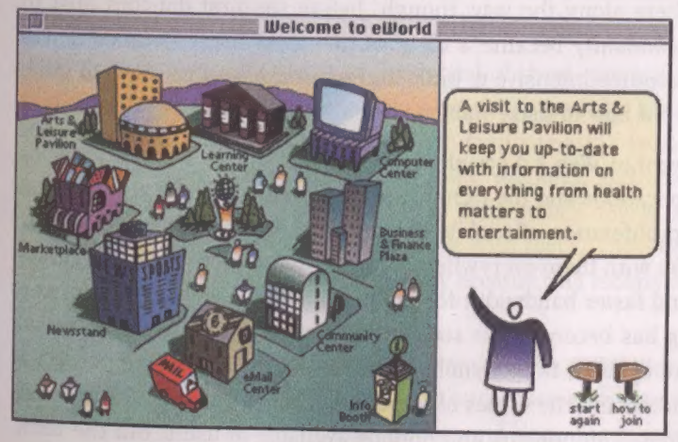


FIGURE 1-1
The very graphical interface of eWorld was the next step up from BBSs and was competing with AOL.

The first example that straddled the line between *community* and what we now call *social networks* was the site SixDegrees.com, which made its debut in 1997 (see Figure 1-2).

FIGURE 1-2
SixDegrees.com was one of the first social networks that connected people and built user profiles.

SixDegrees showcased connections among people, gave users the ability to create and manage their personal profiles, and brought people together based on interests and other features. Sound familiar?

Somewhere along the way, though, before the first dot-com bust in 2000, *community* became a dirty word—most likely because it was overly resource-intensive to build and maintain, and no one had quite figured out how to make money from all that work.

The advent of Web 2.0 paved the way for the second wave of websites and applications and the richer experiences they offer. It also ushered in the proliferation of mobile devices, allowing people to carry their networks with them everywhere they go—more sophisticated technologies and faster bandwidth for the masses. With Web 2.0, social networking has become table stakes. Every experience must have social pieces integrated. In fact, mobile thrives on this, and even enterprises now understand the values of these features. In this phase, social has many more components and options available to users, but the term still generally means features or sites that allow interaction in real or asynchronous time among users. The tools are more robust, storage space is more ample, and more people are online to participate. The increase in online population is a major driving force for the shift to prioritizing these types of features and sites. There is critical mass now. By 2014, 87 percent of Americans, 70 percent of Europeans, and more than 50 percent in the rest of the world were online and participating.

Another key difference between the first cycle of social then and the current cycle today is that the social network—the real relationships with people that we know and care about—is key to the interactions and features. Features are gated based on the degrees of connection between two people. Many of the tools, apps, and websites offer features and functions that support existing offline relationships and behaviors. These places count on each person bringing his personal network into the online experience. The concept of tribes and friends has become more important than ever and has driven the development of many products.

What was ho-hum in 1997 is now the core—for user features as well as opportunities for making money. Additionally, the power of the many, or the wisdom of crowds, is being utilized to exert some control over content creation and self-moderation processes. Companies are learning that successful social experiences shouldn't and can't be overly controlled. They are learning they can take advantage of the crowd to do some of the heavy lifting, which in turn spares them some of the costs. User-generated content has helped many businesses and the participating community to keep things moderated.

The other factor contributing to the spread of these types of features is the expertise of a new generation of users. These folks have grown up with technology and expect it to help facilitate and mediate all of their interactions with friends, colleagues, teachers, and coworkers. They move seamlessly from computer to their mobile device or phone and back, and they want the tools to move with them. They work with technology, they play in technology, they breathe this technology, and it is virtually invisible to them.

In the past few years, as we have been living in the social web, we are seeing tensions with respect to privacy, and concerns regarding permanence versus ephemerality of content and data. Some people are

beginning to question how this information can be archived for future generations, whereas others want guarantees that the words, ideas, and experiences they share have a half-life shorter than the speed at which they are propagated across the Internet. The deeper philosophical, ethical, and cultural questions are now part of the dialogue, and the differing attitudes across the world and demographically affects how these tools and experiences are developed and nurtured.

The terms *community, social media, and social networking* all describe these kinds of tools and experiences. The terms often are used interchangeably, but they provide different views and facets of the same phenomenon.

In a paper published in the *Journal of Computer-Mediated Communication* in 2007, danah boyd, a noted researcher specializing in social-network sites, and Nicole B. Ellison defined social-network sites as "web-based services that allow individuals to:

1. Construct a public or semi-public profile within a bounded system.
2. Articulate a list of other users with whom they share a connection.
3. View and traverse their list of connections and those made by others within the system. The nature and nomenclature of these connections may vary from site to site."

According to Wikipedia, "social media is the use of electronic and Internet tools for the purpose of sharing and discussing information and experiences with other human beings," and it defines social networking as "a platform to build *social networks* or *social relations* among people who share interests, activities, backgrounds, or real-life connections. Most social network services are web based and provide a means for users to interact over the Internet, such as e-mail and instant messaging services." Community is defined as the group of people who utilize these environments and tools.

Well, What About That Social Media? Can You Expand on That?

The term *social media* first appeared on our radar as a way of generalizing what was going on with blogs circa 2002. The combination of blogging and Really Simple Syndication (RSS) (newsfeeds, feedreaders)—sometimes in the same application (as with Dave Winer's Radio Userland software)—enabled a call-and-response, many-to-many conversational ecosystem to arise, become a bubble, calve into many smaller overlapping and distinct subcommunities, and so on.

In that scenario, the blog posts were the media, but then (as now) much blogging involved linking to sources that themselves might come from the traditional, mainstream media (or *MSM*, as some of the political bloggers tend to refer to it) or from other independent voices. Many people online realized that they were consuming much of their media (i.e., news, gossip, video clips, information) through social intermediaries: reading articles when a more prominent blogger linked to them, discovering media fads and memes by following BoingBoing or many other similar trend-tracking sites, and tuning in to the blogs and publications of like-minded people and relying on them to filter the vast, unfathomable information flow for those valuable nuggets of relevancy.

Along the way, the term *social media* began to stand in for Web 2.0, or the Social Web, or social networking, or (now) the experiences epitomized by Facebook and Twitter. Christian called this "the living web" in his last book, *The Power of Many: How the Living Web Is Transforming Politics, Business, and Everyday Life* (Wiley). Technorati tried branding it as the "World Live Web." The idea is that as the Internet in general becomes more *social* (that's the word we've all converged on) and everything is social and social is everywhere, there is an element of it that is read-write that involves people writing and revising and responding to one another, not in a one-to-one or one-to-many fashion, but many-to-many. The problem with using *social media* as a generic term for the entire Internet-enabled social context is that the word "media," already slippery (does it refer to works of creation or to finding relevant news/media items, or to public chatter and commentary, or all of these things?), begins to add nothing to the phrase, and doesn't really address the social graph.

We continue to see a proliferation of social-media marketing experts and gurus online, and their messages range from the sublime (that marketing can truly be turned inside out as a form of customer service, through Cluetrainful engagement† with customers, that is, treating them as human beings through ordinary conversations and public responsiveness), to the mundane (as in the early days of the Internet, every local market has its village explainers), to the ridiculous (a glorified version of spam).

† *The Cluetrain Manifesto* (http://www.cluetrain.com)

O'REILLY®

Designing Social Interfaces

Designers, developers, and entrepreneurs today must grapple with creating social interfaces to foster user interaction and community, but grasping the nuances and the building blocks of the digital social experience is much harder than it appears. Now you have help.

In the second edition of this practical guide, UX design experts Christian Crumlish and Erin Malone share hard-won insights into what works, what doesn't, and why. With more than 100 patterns, design principles, and best practices, you'll learn how to balance opposing forces and grow healthy online communities by co-creating the experience with your users.

- Understand the overarching principles before applying tactical design patterns
- Cultivate healthy participation and rein in misbehaving users
- Learn patterns for adding social components to an existing site
- Encourage users to interact with one another, whether it's one-to-one or many-to-many
- Use a rating system to build a social experience around products or services
- Orchestrate collaborative groups and discover the real power of social networks
- Explore numerous examples of each pattern, with an emphasis on mobile apps
- Learn how to apply social design patterns to enterprise environments

"*Designing Social Interfaces* is a fantastic reference for people who want to find the right design patterns fast. If you need to understand how to connect your users to your product and to each other, this book is perfect for you."

—Laura Klei
author of *UX for Lean Startup*

"Chock-full of visual examples, *Designing Social Interfaces* is a smart and readable reference that will help you apply appropriate design patterns so you don't waste time re-inventing the wheel."

—Kate Rutter
UX Instructor and Visua
Thinking Consultant, Intellet

Christian Crumlish is VP of Product at 7 Cups of Tea (7cups.com). He also served as director of product at CloudOn, director of messaging products at AIM, and curator of the Yahoo! Design Pattern Library.

Erin Malone, Principal with Tangible UX, has over 20 years of experience leading design teams and developing social experiences for a host of startups and large enterprises. She's the former director of the team that produced the Yahoo! Design Pattern Library.

INTERNET APPLICATIONS / SOCIAL WEB

US $49.99 CAN $57.99
ISBN: 978-1-491-91985-9

Twitter: @oreillymedia
facebook.com/oreilly

SECOND EDITION

Designing Social Interfaces

Crumlish & Malone

O'REILLY®

Designing Social Interfaces

PRINCIPLES, PATTERNS, AND PRACTICES FOR IMPROVING THE USER EXPERIENCE

Christian Crumlish & Erin Malone

Contents

The Layperson's Guide to Water Quality is prepared and distributed by the Water Education Foundation as a public information tool. It is part of a series of Layperson's Guides which explore pertinent water issues in an objective, easy-to-understand manner.

Author: J.K. Hartshorn
Editor: Rita Schmidt Sudman

On the Cover:

A clear mountain lake typifies a precious resource we once took for granted — clean, high-quality water.

Photos

State Water Resources Control Board
Marvin Wieben, Jr.
Metropolitan Water Dist. of So. Calif.
Imperial Irrigation District
Calif. Department of Fish and Game
U. S. Bureau of Reclamation

Artwork

State Water Resources Control Board
Cathy Clark-Ryll
Contra Costa Water District

Reprinted 1987

Printed by: Paul Baker Printing

The Water Education Foundation is a non-profit tax-exempt organization. Its mission is to broaden public understanding of current California water resources issues by fairly and accurately presenting information and the views of responsible persons and organizations.

For information on the Foundation's other information and education programs contact:

Water Education Foundation
717 K Street, Suite 517
Sacramento, CA 95814
(916) 444-6240

Introduction

Water ***quantity*** problems have plagued California since the days of its early settlers. Our state's massive water transportation and storage systems are testimony to the fact that California's water supplies quite often do not occur where and when they are needed most. Our water supply is sometimes a case of glut or famine, flood or drought.

But all the water in the world—in the right place, at the right time—won't do a drop of good if it isn't fit to use. True, our domestic water supplies have come a long way since the days when a glass of water might carry with it the threat of cholera or typhoid. But almost daily, the news media carry alarming stories of toxic wastes threatening our ground water supplies. We've learned that our surface supplies aren't immune from exotic-sounding contaminants, either. The very chemical used to destroy disease-causing bacteria in drinking water has given rise to a whole new problem—the formation of substances which might possibly cause cancer. The ***quality*** of our water supplies, once taken largely for granted, is becoming the focus of increasing concern.

Not that water quality and quantity aren't related. We in California place a tremendous burden on our water supplies and have built metropolitan centers supported by imported water. If one of the sources on which we've come to rely were to be rendered unusable, reverberations from this loss could be felt statewide. In coastal areas, overdrafting a ground water basin pulls saline water into the aquifer, polluting the usable supply. The Sacramento-San Joaquin Delta, a cornerstone in the state's water supply and water transportation system, depends on quantities of fresh water pouring in to repel the salty waters of San Francisco Bay and to maintain a quality that makes Delta water suitable for domestic, recreational, agricultural and industrial purposes.

Nor do our water supplies exist in a vacuum. Surface and ground water supplies are surrounded by land and air, making it difficult, if not impossible, to separate factors affecting the quality of one of these resources from the other two.

Over the years—in very recent years, particularly—water quality technology has evolved into a complex subject. Today's water quality technologist must be familiar with many branches of science, among them chemistry, biology, bacteriology and toxicology. As we continue to come up with new chemicals, and as analytical methods are refined and our ability to detect and measure substances in ever-smaller quantities grows, so does the confusing list of compounds which have the potential for impacting our environment: dibromochloropropane (DBCP), trichloroethylene (TCE), perchloroethylene (PCE), polychlorinated biphenyls (PCBs), trihalomethanes (THMs) and a host of others that have yet to turn up.

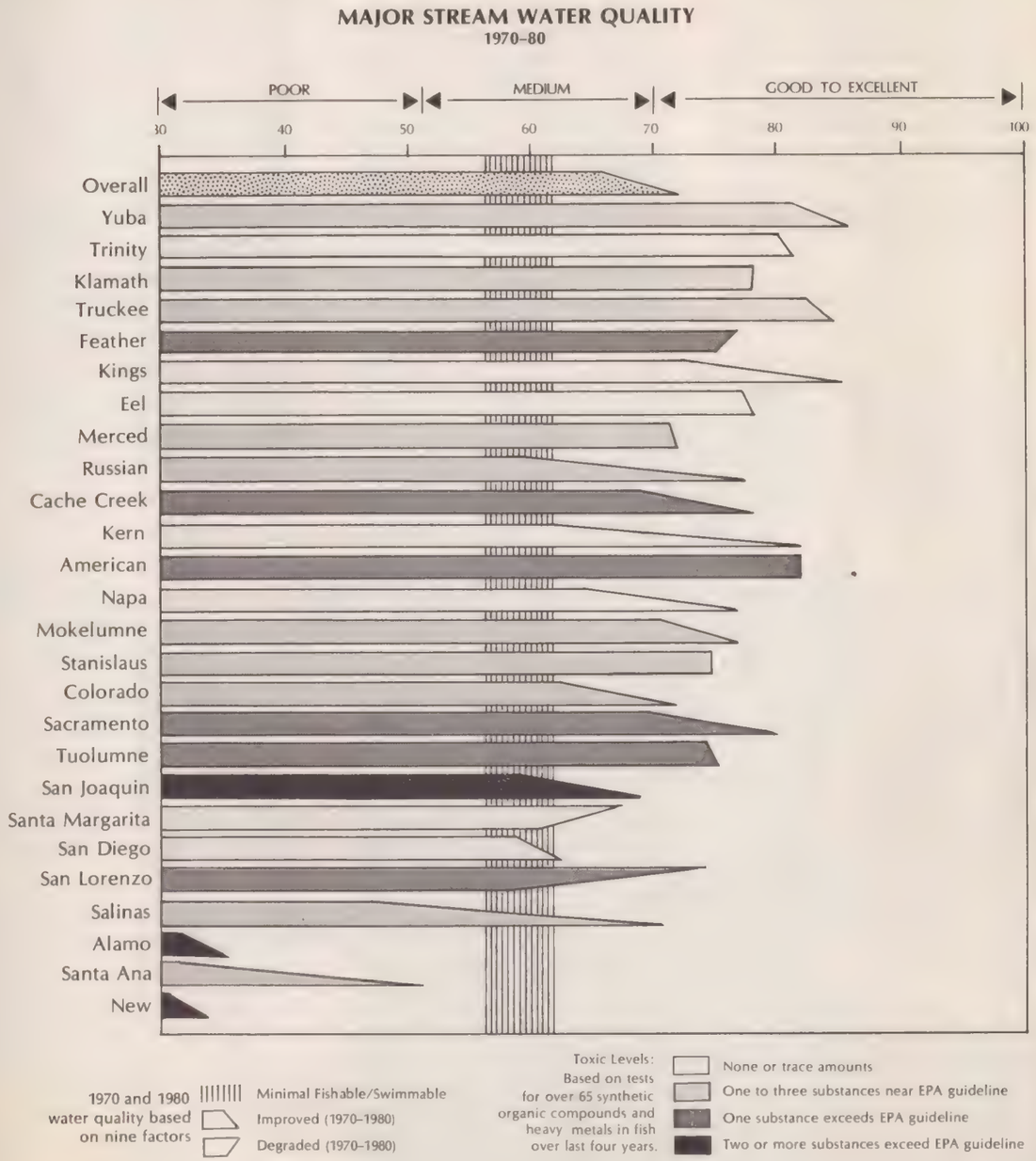
This Layperson's Guide presents a necessarily broad overview of the facts, issues and technology surrounding water quality, water quality considerations of some of the different uses of water and the treatment of water supplies to make them safe for human consumption.

For additional information on related topics, consult the Layperson's Guide to Water Reclamation and the Layperson's Guide to Ground Water.

Other easy-to-understand materials available from Water Education Foundation include guides to San Francisco Bay-Delta, Water Conservation, New Melones Dam, Water Reclamation, and Colorado River.

How Clean Are Our Waters?

3



Background

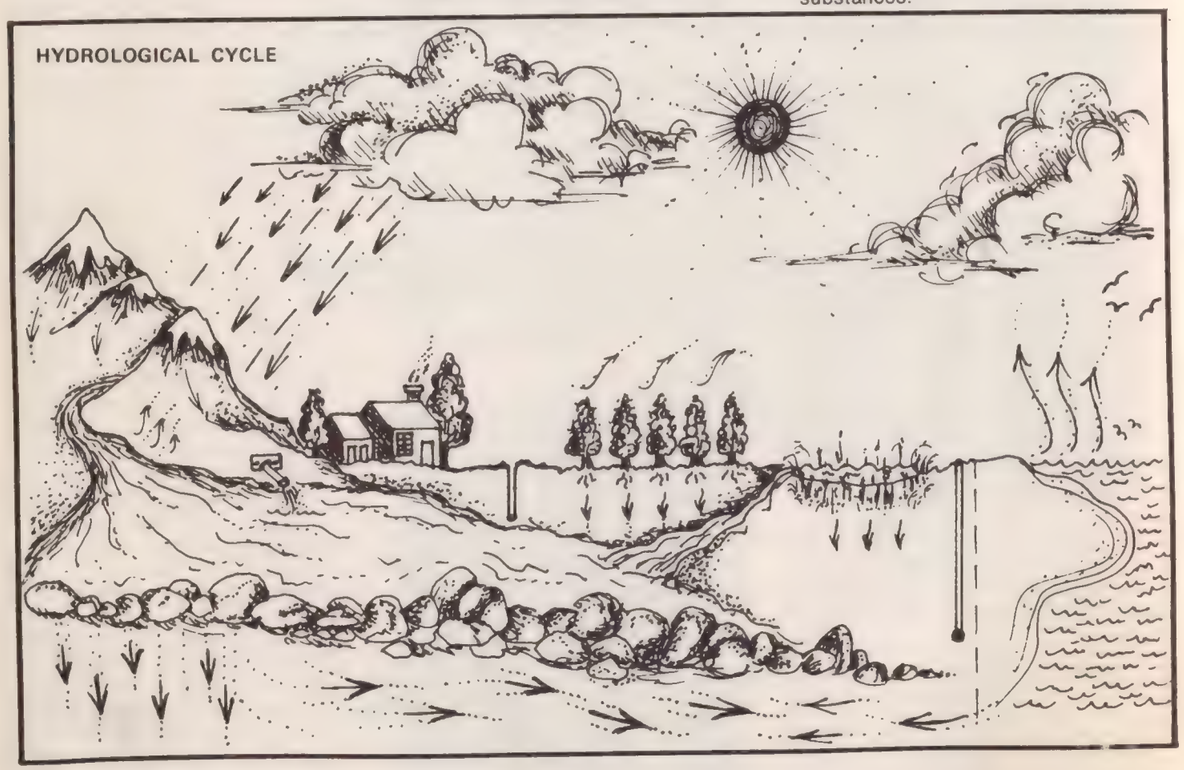
"Water quality" in itself is a neutral term. The uses of water dictate its necessary quality; water which is perfectly suited to growing cotton may have to undergo treatment before it can be used in manufacturing cardboard boxes. Water which is fine for household use may not be for brewing beer. And while we desire "pure" drinking water, water that is too free of dissolved minerals is flat and tasteless.

The hydrologic cycle is nature's own water purification and recycling system. Water is purified through evaporation from bodies of water and transpiration from plant life and is returned from the skies in the form of precipitation—rain, snow, sleet or hail.

But "pure" water—two atoms of hydrogen, one of oxygen and nothing else—can exist but briefly in the cycle. Because water is the "universal solvent," it picks up an assortment of gases, minerals and other substances in its journey through the hydrologic cycle. As water molecules condense in the atmosphere, the droplets collect around tiny particles of dust, smoke and salt and combine with gases, among them atmospheric carbon dioxide, to form weak acids. Once the droplets reach earth, their acidic properties break down rocks and dissolve minerals in the rocks and soils. These minerals include sodium, chlorides (salts containing chlorine and a metal), calcium, iron and magnesium.

Minerals and other substances found in a water supply are also affected by the whole spectrum of natural and manmade materials and activities, such as vegetation, temperature, logging, mining, construction, irrigation, industrial activity and air pollution. Water quality is dynamic, changing from season to season, day to day and even from minute to minute.

Substances found in water may be in suspension (undissolved) or in solution (dissolved in the water). Suspended particles commonly found in raw water supplies include bacteria and other microorganisms, silt and asbestos. Substances dissolved in a water supply include minerals and natural and synthetic organic substances (containing carbon). If one were to take a glass of water apart molecule by molecule, one would find literally thousands of substances.



The constituents of water are commonly expressed in parts per million (ppm) or milligrams per liter (mg/L), equivalent terms. (One part per million is one part of the substance dissolved in one million parts of water.) New instruments allow scientists to discover and quantify substances in even smaller concentrations, expressed in parts per ***billion*** (micrograms per liter, or ug/L) or the incomprehensibly minute parts per ***trillion*** (nanograms per liter, ng/L).

In view of the abuses to which they sometimes are subjected, surface waters have remarkable regenerative capabilities. As long as it contains enough dissolved oxygen, a freeflowing stream can somewhat reduce biodegradable organic pollution, such as sewage.

Bacteria naturally present in the stream utilize oxygen in the breakdown of the organic matter, which they use as food, in a process known as aerobic ("with free oxygen") decomposition. As the stream flows along, oxygen used by the bacteria in this process is replaced through the absorption of oxygen from air into the water.

The key element in this equation is oxygen. Without enough dissolved oxygen, decomposition of organic matter proceeds anaerobically ("without free oxygen"), resulting in the production of noxious gases and sludge and the death of aquatic life.

The amount of dissolved oxygen required by microorganisms in decomposition under aerobic conditions is expressed in terms of biochemical oxygen demand (BOD). The larger the amount of organic waste discharged into a stream, the more dissolved oxygen and time will be needed for decomposition.

Lakes and reservoirs, because of their still waters and exposure to the sun, respond more sensitively to pollution than streams. Their placid waters foster the growth of algae. While some algae serve as food for other aquatic life, large amounts create a cycle of rapid growth, oxygen depletion and decay. Algae proliferate when there are large quantities of nutrients, such as phosphate or nitrogen, entering the lake; when this occurs, the lake is called ***eutrophic*** ("well-nourished") and the process is known as eutrophication. Eutrophication is a natural process, but it can be accelerated by human activities.

For regulatory purposes, pollution or contamination is grouped into two broad categories. There is ***point source pollution***, coming from a single, identifiable source such as discharge from a sewage treatment plant or factory, and ***non-point source pollution***, which isn't concentrated at a single site and can't be as readily controlled. A major non-point source is the runoff from streets and other surfaces of urban areas. Other examples include agricultural runoff, mining, timber harvesting, leakage from hazardous waste sites, natural erosion of streambanks and seawater intrusion.

The placid waters of lakes foster algae growth

Regulation

Attempts to exert some control over water pollution began at the end of the nineteenth century, with programs aimed at controlling waterborne infectious diseases directly related to the disposal of human wastes into water sources. In 1901 the U. S. Public Health Service was created and in 1912 its authority broadened to include control over pollution of surface waters. For the next 36 years, though, enforcement of water quality standards was largely up to the individual states.

California has long been a forerunner in water pollution control. The effort was stepped up following several sewage and industrial waste pollution incidents in the 1940s, leading to the passage of the Dickey Water Pollution Act of 1949. This act created nine regional water pollution control boards to establish and enforce water quality standards, under the direction of a State Water Quality Control Board—forerunners of today's Regional Water Quality Control Boards and State Water Resources Control Board.

On the federal level, recognition of the growing problem of water pollution resulted in the passage of the federal Water Pollution Control Act of 1948 and increasingly stronger amendments aimed at controlling the discharge of pollutants into waters and providing for governmental construction grants for waste treatment facilities.

The "Clean Water Act" for cleaning up polluted waters (Federal Water Pollution Control Act Amendments of 1972 and 1977, Public Law 92-500) represented a major commitment to restoring and maintaining national waters by setting a date by which waters must be restored to "fishable, swimmable" quality and encouraging the reclamation and reuse of chemicals.

Since the act's passage and reauthorization in 1977, Congress has appropriated more than $37 billion for grants to aid construction of pollution control facilities.

The Clean Water Act is administered by the U.S. Environmental Protection Agency (EPA), beset in recent times by controversy and budget cuts. It comes up for review every five years and in 1983 was undergoing reassessment. Not everyone agrees that the benefits achieved by the act have been commensurate with the costs. While environmental groups do not want any relaxation of goals and standards, there is a recognition that practical limits exist on available funds within the nation's economy and tradeoffs may be necessary.

Two years before the passage of the federal Clean Water Act, Californians made pollution cleanup a goal by overwhelming approval of a $250 million bond issue for the construction of waste treatment facilities. In 1974, a second $250 million bond issue was passed, and from 1975 to 1977 officials put California's Clean Water Grant funds to work, under the control of the State Water Resources Control Board (Water Board).

In 1978, while showing their desire to put a rein on spending by passing Proposition 13, the property tax initiative, voters at the same time signalled their commitment to the Clean Water Grant program by approving a $350 million bond issue.

The state Water Board notes that efforts to control point source pollution have been "greatly successful," with some $4 billion spent in the past decade on wastewater treatment facilities, although some municipal wastewater plants in major metropolitan areas have yet to be completed.

Water pollution control facility at Point Loma, San Diego

On the state level, California's water rights and water pollution control are the responsibilities of the State Water Resources Control Board in Sacramento, and nine Regional Water Quality Control Boards. This responsibility was assigned to the Water Board in 1969 with the passage of the Porter-Cologne Water Quality Act, which increased the protection of the state's streams.

The nine regional boards were charged by the act to formulate "Basin Plans" for their hydrologic areas and to set standards and enforceable limits for waste dischargers who obtain requirements from the regional boards. The regional boards also inspect facilities and perform sampling to ensure compliance with the standards set for each region.

Many other federal and state agencies work with the Water Board to monitor water quality, among them the U.S. EPA, U.S. Bureau of Reclamation, U.S. Geological Survey and the state departments of Water Resources, Fish and Game and Health Services.

A legal hazardous waste site near Benicia, CA . . .

. . . And an illegal dump site

Hazardous Wastes

The need to monitor hazardous chemicals has only recently come into the public eye, with the realization that hazardous substances improperly used or disposed of on land can affect water quality.

In 1976, Congress passed the Resource Conservation and Recovery Act (RCRA), madating the EPA to establish safety regulations for disposal of hazardous wastes, penalties for violators of the regulations and a national system for tracking hazardous wastes "from cradle to grave"—from their manufacture to their disposal.

In 1980, the controversial $1.6 billion federal "superfund" was established to accelerate the cleanup of the most hazardous sites. More than 400 sites were targeted nationwide, 21 of them in California.

According to EPA's 1981 estimates, there are approximately 5,500 known operators engaged in generating, transporting, storing, treating and/or disposing of hazardous wastes in California.

California's program predates the RCRA and is the responsibility of the state Department of Health Services' Hazardous Waste Management Branch. One responsibility of the branch is to direct the state's own "superfund," a 10-year, $100 million program funded by taxes collected from waste generators, which targets 60 sites around the state, including those on the federal "superfund" list, for immediate action.

While the cleanup of sewage and industrial discharges has greatly reduced point source pollution, the Water Board points out that controlling toxic chemicals is a "major challenge" of the 80s.

Water Quality Requirements

Agriculture

Just mention "salt" and the average person no doubt will picture the table variety—sodium chloride. To the farmer, however, "salts" and "salinity" generally refer to total dissolved solids (TDS), or all the salts dissolved in irrigation water, of which sodium and chlorides are just two. Other combinations include calcium and magnesium chloride; calcium, magnesium and sodium sulfate; and calcium, magnesium and sodium bicarbonate. Small amounts of potassium, nitrate and carbonate also are present.

Without adequate drainage, the salts carried by irrigation water accumulate in the soil, causing leaf burn, stunted plant growth and loss of productivity. Historically, salinity buildup in the soils of Middle Eastern countries has destroyed agriculture and led to the downfall of civilizations. Salinity is a problem faced by farmers in some of the state's richest agricultural areas and is being met with varying degrees of success.

The concentrations of salts in a surface water supply vary with the flow. They are higher when the flow is low, and vice versa. Surface waters in the state range from those low in salinity, such as waters flowing out of the Sierra Nevada and Northern California, to the more saline Colorado River. The salinity of ground water supplies varies as well.

While total dissolved solids are expressed in terms of their concentrations in water, in parts per million (ppm), the salt tolerance of crops is expressed in terms of the electrical conductivity (EC) of the soil solution using measurements known as millimhos per centimeter, or an equivalent term now coming into common usage, deciSiemens per meter.

Electrical conductivity is a measure of the osmotic pressure of the soil solution surrounding plant roots; osmotic pressure influences the amount of water the roots can extract from the soil. Electrical conductivity is measured by determining the electrical resistance to alternating electrical current run through a sample of soil solution. Electrical conductivity varies directly with total dissolved solids.

The salt tolerance of crops varies tremendously, ranging from salt-sensitive strawberries to cotton, which can tolerate about 10 times more salinity in the root zone than strawberries before yields decline. Some crops, such as citrus fruits, are particularly sensitive to the element boron, while stone fruits (peaches, avocadoes, grapes) are sensitive to sodium and chloride.

Leaching—applying sufficient amounts of irrigation water to flush salts out of the soil—is very important to prevent salts from concentrating. Farmers can cope with saline waters and soils in other ways, too. They can change the way they apply irrigation water to crops, switching from furrow irrigation, which brings water into the root zone from the side, to sprinkling, which delivers water from the top and allows for better percolation into the soil. Farmers may choose to change their cropping patterns and plant salt-tolerant crops, such as barley or cotton, rather than more sensitive crops, or change bedding shapes or frequency of irrigation. A wide range of factors—soils, climate, economics—influence what a farmer can successfully grow, however.

Underground drainage pipe is laid in an Imperial Valley field

In the San Joaquin Valley, agricultural productivity is threatened by saline high water tables, resulting in an estimated annual loss in net farm income of $31.2 million. State and federal water supply experts propose a central drainage disposal facility to carry away saline drainage waters and spur the construction of on-farm subsurface drainage systems and drainage collector systems.

Hundreds of miles to the south, in the Imperial Valley, nearly a half-million acres of productive farmland are irrigated with Colorado River water. The present salinity of the Colorado in terms of total dissolved solids is high (800 ppm and higher) and has increased over the years due to the return of drainage water from upstream irrigation projects, and to a lesser degree, to evaporation losses from the large upstream storage reservoirs. Future increases in water-using developments in the Upper Colorado River Basin could result in further increases in the river's salinity.

In recent years, plans for salinity control in the river have been developed by the Colorado River salinity Control Forum, a seven-state organization. Within this control effort, the Colorado River Basin Regional Water Quality control Board is monitoring developments which could result in additional salt loads on the river.

Over the past several decades, though, the increased use of subsurface drainage pipe (historically called "tile") in the Imperial Valley has resulted in a positive salt balance—more salts leached from the soils and carried out of the valley than are brought in by the Colorado River. The pipe is periodically spaced in fields at an average depth of six feet. Irrigation water percolates into the ground, picks up salts in the soil and finds its way into the pipe through small perforations or joints. The water then flows through the pipe and on to a drainage outlet, into a surface drainage ditch and into the Salton Sea.

Industry

As industrial processes and products differ, so do needs for varying qualities of water. While we may think of drinking water as the purest water available, it may not be pure enough for certain industrial processes and may need specialized treatment by the industry itself to make it usable.

At Anheuser Busch's Van Nuys brewery, water is used to maintain strict cleanliness standards and also for the brewing of beer. All water coming into the plant must be run through diatomaceous earth filters, while that used in the brewing process must be put through an activated carbon filter to remove chlorine, the brewery's main concern.

At the Technicolor Laboratories in North Hollywood, hardness causes headaches. Most of the water used in the processing of film must be extremely soft (low in magnesium and calcium) and is run through a softener much like a home softening unit. Water that is too hard forms calcium deposits on the film that show up when the film is projected. In addition, water must be deionized for the film's final stabilizing bath, so colors will remain true, and for use in the plant's quality control laboratory.

High chloride levels worry the paperboard manufacturing industry. Louisiana Pacific Corporation, which operates a large corrugated paperboard plant in Antioch, in the Sacramento-San Joaquin Delta, pretreats raw water coming into the plant with alum and runs it through an ion-exchange unit to lower chloride levels.

High-technology industries engaged in the manufacture of computer components must put the water they use through the most extensive treatment of all. This is particularly true in the manufacture of integrated circuits, according to sources at Hewlett-Packard. Water used to rinse the circuits during the manufacturing process is run through a number of processes, including reverse osmosis (forcing water through a membrane with tiny pores, to separate the water molecules from the salt molecules), deionization, filtration through extremely fine filters and exposure to ultra-violet light (to kill bacteria).

Generally, the water used for processing California's bountiful harvest of fruits and vegetables does not have to undergo elaborate treatment, although some of it does receive additional chlorination at the plant, and water used in a plant's boilers must undergo the same demineralization as in other industries.

Water used in canning must meet health standards and is usually obtained from municipal supplies or from wells, according to a spokesperson for the National Food Processors Association.

Industrial water use requirements vary with each industry. Pictured is one of the many industries in the Delta.

Fish and Aquatic Life

Aquatic biologists still remember the days when sewage pollution routinely caused massive fish kills. Fish are dependent on oxygen in water, and when the oxygen diminishes because of excessive amounts of untreated sewage, sensitive fish such as trout die or move on. They may be replaced by hardier, less desirable non-game fish, such as carp.

While point source discharges, such as sewage, are no longer the problem they once were, non-point source discharges are a cause of concern.

The state Department of Fish and Game collects and tests aquatic life from rivers and lakes, reporting results to the State Water Resources Control Board. Fish and Game scientists test for concentrations of heavy metals, such as mercury, which can be toxic to fish, as well as for organic compounds, such as pesticides and herbicides.

Although it was banned a decade ago, the pesticide DDT still shows up in fish samples taken from many rivers in California. While the levels are seldom high enough to violate health standards for either fish or humans, no one knows how long DDT will persist in soils and waters. Mercury, chlordane (another pesticide) and toxaphene (yet another pesticide now banned) also have shown up in some fish samples.

A few years ago, PCBs (polychlorinated biphenyls, used in electrical transformers) were discovered in fish taken from the Feather River. The source was pinpointed to oils, applied to roads to control dust, and the problem was corrected by paving the roads and sealing in the contaminated area.

Fish also are sensitive to thermal pollution, elevated temperatures caused by industrial and power plant discharges. In 1975, an EPA-approved Thermal Plan was adopted in California, setting temperature limits for discharges. Thermal standards are being met, according to the Department of Fish and Game.

In addition to the discharge of wastes, many fish are lost due to water diversions. Power plants, water supply intakes and agricultural water diversions can draw in fish along with the needed water supplies. This problem has yet to be solved, but progress is being made through the use of fish screens and of the federal Clean Water Act and Fish and Game laws.

Domestic Water

Drinking water supplies, understandably, are an area of great concern. All domestic water served by water agencies is subject to the regulations and approval of the California Department of Health Services. In most cases water must undergo some form of treatment to meet health standards, but even treated water has come under closer scrutiny in recent years.

The primary objective in treating water for drinking or other domestic uses is to remove or destroy all harmful or potentially harmful microorganisms and chemical substances which might be present in the raw water. Also of interest, but from an aesthetic, rather than health, standpoint, is the removal of suspended or dissolved substances which make water look, taste or smell "funny."

Early man no doubt used his senses of sight, smell and taste to judge the quality of the water he drank. Early attempts at water treatment—straining, boiling or allowing sediments to settle out—were aimed at improving the aesthetics of water supplies, but had some unrecognized health benefits as well.

It was not until the mid-1800s that scientists began to suspect that water might carry disease-causing microorganisms, not always detectable by taste or appearance.

Chlorination—adding chlorine to water—was first used in the U.S. in 1908 to destroy potentially harmful bacteria in drinking water supplies, and its eventual widespread use virtually wiped out water-borne diseases such as typhoid and cholera which once were epidemic.

The first drinking water quality standards in the nation were the U.S. Public Health Service Standards for Drinking Water (1914), which specified bacteriological, but not physical or chemical requirements for water. In 1925, copper, lead and zinc were added to the list, and the list was further expanded in 1946 and 1962.

A nationwide survey conducted in the late 1960s, however, showed the need for further revisions, and in 1970 the authority to establish, revise and enforce drinking water standards was transferred to the U.S. Environmental Protection Agency.

In 1974, Congress passed the Safe Drinking Water Act (Public Law 93-523), the goal of which was to provide for safe drinking water supplies by establishing and enforcing maximum contaminant levels (MCLs) for various compounds in water. Carrying out the act's responsibility for establishing national standards fell under the jurisdiction of the EPA. The EPA allowed states which qualified to monitor and enforce the program.

In December of 1975 EPA put into effect National Interim Primary Drinking Water Regulations which included, among other things, certain inorganic substances (such as arsenic, barium, lead and mercury), some insecticides and herbicides, turbidity (clarity), bacteria and radiation. In 1979, it adopted additional interim standards covering trihalomethanes (THMs), a group of organics found in trace amounts in water following chlorination.

Enforcement of drinking water standards in California falls under the Department of Health Services' Sanitary Engineering Branch, which has adopted standards similar to EPA's. These standards are the Domestic Water Quality and Monitoring Regulations, Title 22, California Administrative Code.

Under Title 22, public water suppliers are required to routinely perform tests for the substances specified in the regulations and to report test results to the Department of Health Services.

While not all bacteria are harmful, bacteriological tests provide a sensitive indicator and water suppliers are required to monitor at sample points chosen throughout their entire water systems. The number of required samples ranges from one per month for very small systems to as many as 125 per week for a larger system. Because testing for pathogenic (disease-causing) bacteria is an expensive, time-consuming task, treated water suppliers routinely test for coliform bacteria. These bacteria inhabit human and animal intestines, and although they are not directly responsible for disease, their presence and survival serve to indicate the potential presence of pathogenic bacteria and sewage pollution.

Turbidity, or clarity, of water, once a secondary (aesthetic) consideration, is now a primary regulation. The tiny particles suspended in water are measured in turbidity units (TU), with lower values indicating clearer, or less turbid, water. An importance is placed on turbidity because particles can shield bacteria and allow them to escape disinfection.

Should a water supplier exceed the bacteriological quality limits or maximum contaminant levels for turbidity, organic or inorganic chemicals or radioactivity specified in the regulations, the supplier is required to notify not only the Department of Health Services but also the customers it serves.

Small water suppliers which couldn't otherwise comply with drinking water regulations have been given assistance under California's Safe Drinking Water Bond Act of 1976, administered by the Department of Water Resources. This act made $175 million in loans and grants available for upgrading treatment facilities.

Treatment Processes

The extent to which drinking water must be treated depends on the quality of the raw water.

Some water supplies, such as ground water, may require only disinfection to bring them into compliance with health standards. Other waters coming from sources exposed to significant recreational activities or potential sewage contamination must receive extensive treatment. Treatment plants must be designed on a case-by-case basis, and all processes must be approved by the Department of Health Services.

Conventional Treatment

Conventional, full water treatment consists of the following steps:

The water is put through chemical ***coagulation*** and ***flocculation***, processes involving addition of a coagulant (aluminum sulfate, alum, is commonly used, as are polymers) to assist in the removal of turbidity. Rapidly mixed into the water, alum alters the electrical charges surrounding the suspended particles to make them attractive to each other; they coagulate, or clump together, into larger particles known as ***floc***. Polymers act in a similar way to entrap suspended particles. In flocculation, the water is gently agitated so the floc particles will collide with each other and entrap other suspended particles in the water to form still heavier particles.

During the ***sedimentation*** step, the flocculated water flows slowly through a basin or tank to allow the heavy floc particles to be removed by settling to the bottom.

Filtration is a "polishing" process, which removes additional small, light particles that do not settle, and aids the destruction of bacteria by disinfection. Filter media commonly used include sand and inert materials such as anthracite coal particles, or a combination of these materials.

Disinfection is the most important process for municipal water supplies; this step destroys potentially harmful bacteria. Until very recently the term "chlorination" was synonymous with disinfection, but today, some treatment plants are changing to alternate methods of disinfection, such as different forms of chlorine, or to ozone.

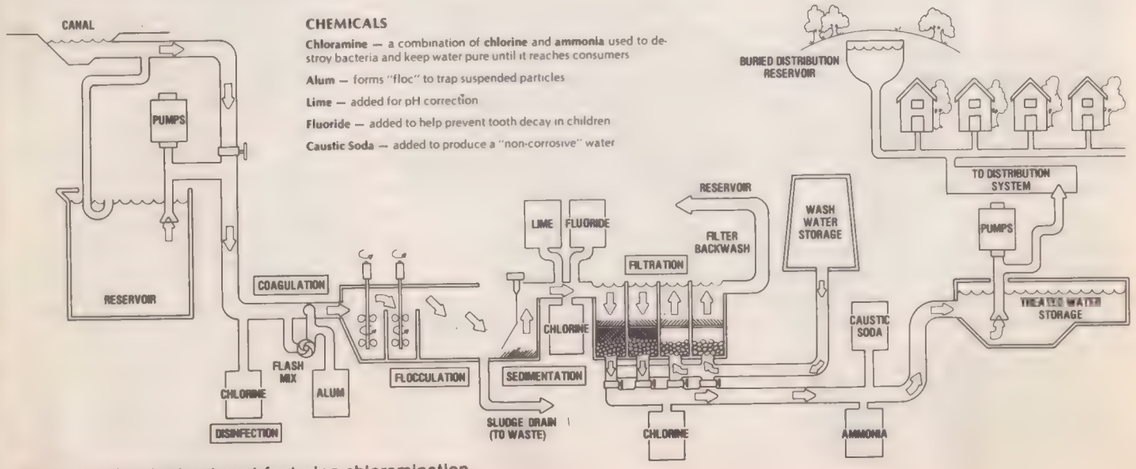
Aeration is sometimes used to remove dissolved gases in water by bringing water droplets into contact with air. It is performed mainly to improve the aesthetics of water and to reduce corrosion.

While not exactly a treatment process, ***fluoridation*** of drinking water to help prevent tooth decay in children has become an established—though still controversial—public health practice. Opponents of fluoridation argue that this "medication" should not be forced on consumers, but the argument has not held up in court. Others claim it is too costly, since only a small percentage of domestic water is actually used for drinking.

Other chemicals may be added to water during the course of treatment. Caustic soda and lime may be added to control the corrosiveness of water and to adjust the water's pH.

Direct Filtration

Direct filtration treatment plants differ from conventional plants in that the sedimentation process is eliminated, although the other treatment steps—coagulation/flocculation, filtration and disinfection—are retained. While this type of treatment is acceptable for raw water with low turbidity and cannot be applied to all raw water, it can save as much as one-third of the construction costs of building a treatment facility.

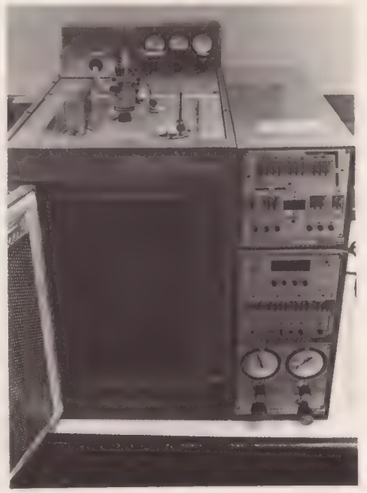


Conventional water treatment featuring chloramination

Health Effects 13

In an age when it seems as though just about everything is suspected of causing cancer, it is hardly surprising that suspected carcinogens are turning up in water supplies.

Historically, there has not been a great deal of interest in these trace substances because up to now there was no easy way of detecting them in very low concentrations. In addition, our knowledge had to advance to a point where we now realize that even very minute quantities of certain substances could pose potential risks over the long term. But while more and more substances appear in the environment, including our food and water supplies, we still are a long way from understanding what, if any, their long-term health effects on humans might be.



Gas chromatograph used for trace organic analysis

Trace Organics

Over the past decade, concern over trace substances in water supplies has focused around synthetic organic chemicals, which have been described as "ubiquitous" in the environment. While hundreds of compounds exist, we either do not know or do not understand the possible health effects of these compounds, some of which are detectable in amounts of micrograms per liter—or less.

One of the sophisticated techniques which makes detection and measurement possible is gas chromatography/mass spectrometry (GC/MS), which can separate the myriad of organics present in a water sample into discreet units by passing a sample through a column containing material which causes the substances to separate. These individual compounds are then "shot" by a beam of electrons, exploding them into fragments. Each explosion pattern is a "fingerprint" of a separate organic compound.

GC/MS units cost hundreds of thousands of dollars, putting them out of the reach of most water suppliers and laboratories, and require highly-specialized and trained professional chemists for proper operation and production of accurate analyses.

Other sophisticated and expensive instruments are now found in larger utility labs. Units such as gas chromatographs, used in analyzing for trihalomethanes, are becoming more common in utility laboratories, and many medium to large utilities use atomic absorption spectrophotometer units to analyze for trace metals. These instruments also must be operated by trained professionals.

Ground Water Contamination

Up until recently, ground water, which accounts for about 40 percent of the water used in California, was considered inviolate. It was assumed that the soil would filter out contaminants, and incidents of ground water contamination were looked upon as isolated occurrences caused by the misuse of chemicals. Ground water contamination was also viewed as something that "happened in other states," such as New Jersey, or New York, with its infamous Love Canal. It wasn't something that happened in California.

Unfortunately, we've learned that soil does not filter out all contaminants. Non-point source discharges, such as from improper surface or underground disposal of hazardous wastes, slowly but surely percolate into ground water supplies. Such contamination may have been going on for years, as ground water moves slowly and it may take decades for contaminants to migrate into wells; only recently have scientists developed the laboratory capability and begun looking for it. And since ground water moves very slowly and lacks some of the natural self-cleaning mechanisms of flowing surface water (like aeration), once it is contaminated it is likely to remain so for decades, perhaps even for centuries.

Trace Organics

Recent investigations have found that ground water contamination by synthetic organics is most common in urban and industrial areas where the chemicals are used or stored. Trichloroethylene (TCE) and perchloroethylene (PCE) are two synthetic organics which have found their way into ground water supplies in several parts of the state. TCE, a suspected carcinogen, is used as an industrial solvent; PCE is another industrial chemical used in degreasing, dry cleaning, aerosols and pesticides.

Another synthetic chemical causing concern is dibromochloropropane (DBCP), widely used by farmers as a nematocide, or pesticide against nematodes, the tiny worms which attack tree and vine roots. The chemical was banned in 1977 after it was discovered to cause sterility in men and cancers in laboratory animals, but residue from past years of use and storage has been seeping into wells in the San Joaquin Valley.

While the jury is still out on the human health effects of many man-made chemicals, action levels, or suggested safe levels have been formulated by the Department of Health Services for some of them. These levels are extremely low; 5 parts per billion (ppb) for TCE; 4 ppb for PCE and 1 ppb for DBCP. If a water supplier exceeds the action level for a chemical, it must blend its contaminated ground water supplies with other waters to meet the guidelines or treat the water until the organic compound concentration levels are reduced below the guidelines.

Trihalomethanes

Within the past decade, great interest has been generated by the discovery of trace organics known as trihalomethanes, or THMs, formed when chlorine used in the disinfection of drinking water combines with naturally-occurring organic substances present to some degree in all surface water supplies.

These organic substances, such as humic and fulvic acids, are referred to as THM "precursors" and may come from decaying vegetation and soils in watersheds. The most common THM, chloroform (once an additive to toothpaste and cough medicine) has caused cancers in laboratory animals.

Concern over these potentially carcinogenic substances led the EPA, in 1979, to establish a maximum contaminant level of 0.10 milligrams per liter, based on a running quarterly average, for water systems supplying 75,000 or more people; the regulation went into effect in 1981. By the end of 1983, water systems serving more than 10,000 people also will be required to meet the 0.10 mg/L requirement.

THM Control

While methods of controlling THM precursors or THMs ***after*** they have been formed have received some study, the methods receiving the most attention from utilities center around altering disinfection processes to prevent the formation of THMs, at the same time adequately disinfecting water and maintaining a residual amount of disinfectant in the distribution system. Alternate disinfection methods include the use of chloramines, ozone or chlorine dioxide. Considerations which must be taken into account are the alternative's effectiveness, non-toxicity to humans and its cost.

The control of THMs is of particular interest to agencies which treat surface water. In 1981, Contra Costa Water District in Northern California and the City of San Diego received permission from the Department of Health Services to use a chlorine-ammonia combination called ***chloramine***, to disinfect the water served their treated water customers. Other water suppliers have followed suit and have either initiated or are studying chloramination of water supplies; some of these include Santa Clara Valley Water District, Metropolitan Water District of Southern California, Alameda County Water District and the City of San Francisco.

Besides the fact that chloramination does not form THMs, chloramines persist longer and can be as effective as chlorine in reducing bacteria in water distribution systems. For this reason, Oakland-based East Bay Municipal Utility District in 1983 launched a plant-scale pilot study which could lead to the use of chloramines at all EBMUD filter plants.

Although the Los Angeles Department of Water and Power does not exceed the THM limit, the utility plans to use ***ozone***, rather than chlorine, as the disinfectant at a new direct filtration plant now under construction. Ozone is an excellent disinfectant and according to DWP, it will help reduce already low THM levels.

Organics Removal

It is possible to remove THMs and other trace organics once they are in a water supply, but the technology is more expensive.

One method of removal, known as ***air stripping*** costs about twice as much as changing the method of disinfectant to control THMs. It involves passing large volumes of air through contaminated water; volatile organics such as THMs tend to transfer from the water to the air. Packed tower aeration involves trickling water down through 10 to 40 foot towers filled with porous material, at the same time forcing air up through the tower. The water separates into an extremely thin film, allowing for maximum contact between the water's surface and air, much more than with normal aeration. Because air stripping releases organics into the air, though, some people worry about transferring pollutants from one resource to another.

A much more expensive method of removing some organics involves filtering water through ***granular activated carbon*** (GAC), carbon pieces which have been specially treated to have maximized surface area for trapping organics.

GAC filters can't be cleaned by backwashing (flushing) the way other filters are. When GAC becomes saturated it must be removed and regenerated by heating it to more than 1000° Farenheit. The time between regenerations depends on the organics being removed and can range from several months to a year for non-volatile synthetic organics such as PCBs and pesticides, to every few weeks for THMs.

This regeneration process makes granular activated carbon cost anywhere from several times to 100 times as much as air stripping.

Asbestos

Fibrous minerals known as asbestos are commonly found in air and water throughout the state. Asbestos was mined on the western slopes of the Sierra Nevada and is found in the serpentine rock of California watersheds. Even in large quantities, the microscopic fibers are invisible to the naked eye.

Asbestos is a known occupational hazard when workers are subjected to high levels over prolonged periods of time. While inhaled asbestos is linked to diseases of the lungs, there is no confirmed medical relationship between asbestos fibers in drinking water and cancer. It appears there may be some mechanism scientists don't yet understand which causes asbestos to be "deactivated" in the stomach, some water quality experts suggest.

Efficient filtration combined with full conventional water treatment can eliminate more than 99 percent of the asbestos fibers in raw water.

Parasites

Just because a water is clear, cold and free of objectionable odors or tastes doesn't necessarily mean it is safe to drink. Even sparkling mountain streams can hide microscopic animals known as ***giardia lablia***, which enter the stream via fecal contamination. The symptoms of giardiasis, the serious infection caused by the parasite, often appear several weeks after the unsuspecting backpacker or camper has drunk the contaminated water. Precautionary measures which can be taken by those venturing into the wilds include carrying in or boiling drinking water.

Summary

As we've seen, water users and suppliers have a number of methods at their disposal for dealing with water quality problems and rendering water suitable to their purposes.

Of course, these solutions don't come without price tags. It is far from inexpensive for a farmer to lay drainage pipe in his field, or for an industry to pretreat the water it uses, or for a municipality or water district to ensure the safety of the water it serves its customers.

Cleaning up hazardous wastes, too, will cost us millions of dollars. We'll probably never know how long some of these chemical substances have contaminated our environment—since scientists only recently developed the capability to detect and measure them and began looking for them—and it no doubt will be years and millions of research dollars later before we really know more about their human health effects.

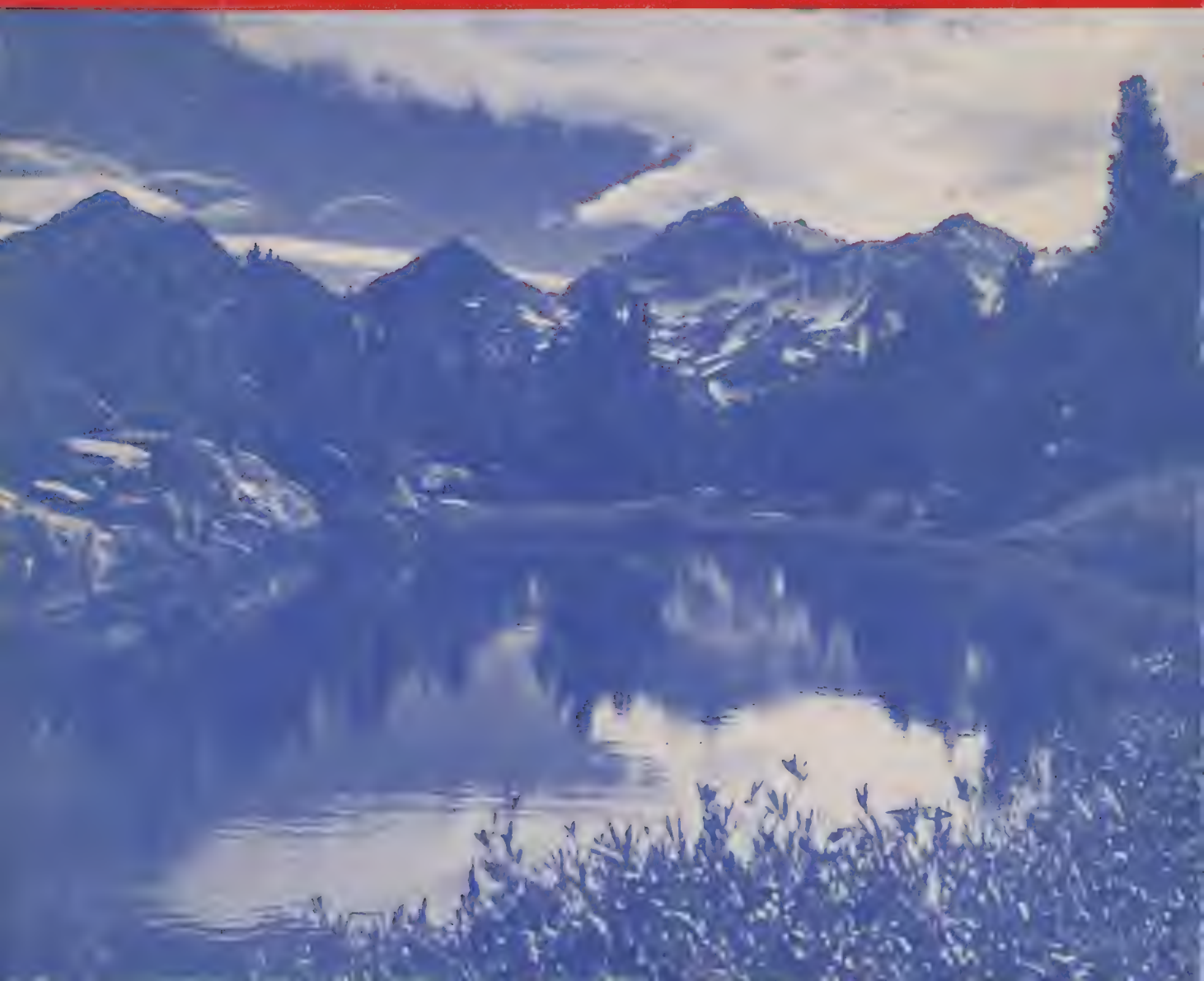
While the technology does exist, it still is not economically feasible for water suppliers to remove some of the impurities from domestic water. Sodium, which has become anathema to persons with high blood pressure, is one example; luckily, drinking water contributes little sodium to the average diet.

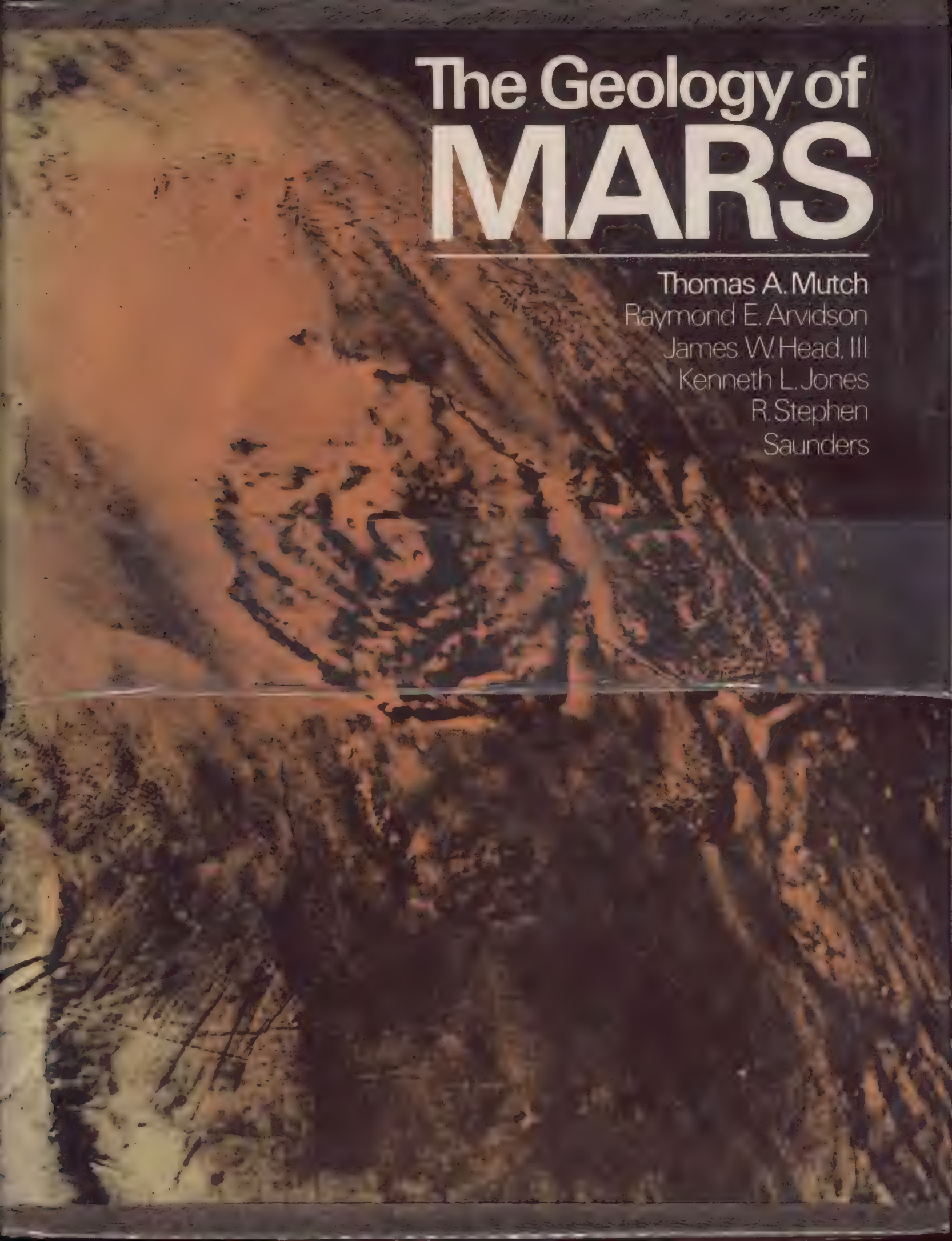
Most important to the consumer, thanks to effective methods of water treatment we no longer need fear death from typhoid, cholera or dysentery. While our waters are not free from the types of quality problems affecting other parts of the country, we in California are pretty fortunate, according to a Department of Health Services expert. California's long history of water quality and pollution control programs have helped us maintain a quality of water that's generally "better than any other in the country."

In light of the problems which have cropped up in recent years and the new ones that are bound to arise, keeping it that way is certain to be a major challenge of the future.

Layperson's Guide Water Quality

Prepared by the Water Education Foundation





The Geology of MARS

Thomas A. Mutch
Raymond E. Arvidson
James W. Head, III
Kenneth L. Jones
R. Stephen
Saunders

sediment superficially resemble terrestrial sand dunes. However, the sediment is finer grained than sand, and the "dunes" have irregular shapes. In part, they appear to be drifts, stabilized by the presence of large rocks. The lineations within "dunes" are internal stratification, revealed by an episode of deflation that followed the initial period of dune accretion. The meteorlogy boom is visible in the near-field

taken with camera 2 on Viking Lander 1. The same field of view appears in the middle left of figure S.10. Several isolated dunes are visible in the midfield. An area of fractured bedrock is in the right midfield, atop a gentle ridge

The Smithsonian Book of

Mars

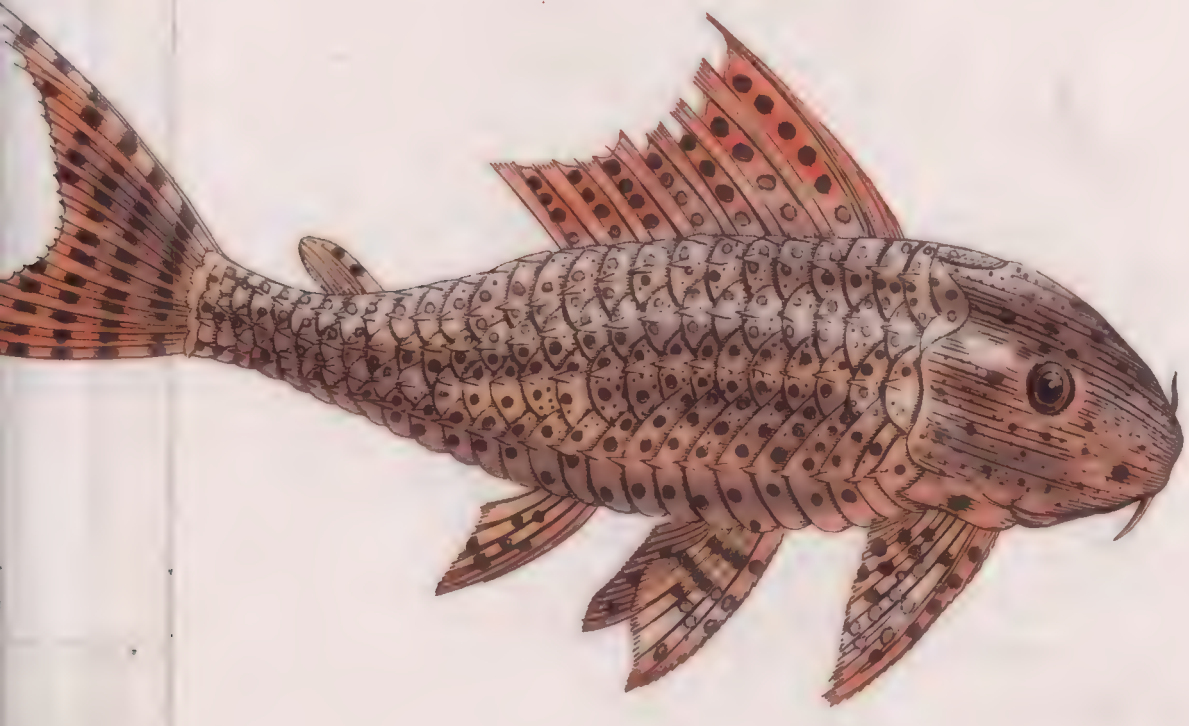
Joseph M. Boyce

O'REILLY®

ST

Container Security

Fundamental Technology Concepts that Protect Containerized Applications



Liz Rice

O'REILLY®

Container Security

To facilitate scalability and resilience, many organizations now run applications in cloud native environments using containers and orchestration. But how do you know if the deployment is secure? This practical book examines key underlying technologies to help developers, operators, and security professionals assess security risks and determine appropriate solutions.

Author Liz Rice, VP of open source engineering at Aqua Security, looks at how the building blocks commonly used in container-based systems are constructed in Linux. You'll understand what's happening when you deploy containers and learn how to assess potential security risks that could affect your deployments. If you run container applications with *kubectl* or *docker* and use Linux command-line tools such as *ps* and *grep*, you're ready to get started.

- Explore attack vectors that affect container deployments
- Dive into the Linux constructs that underpin containers
- Examine measures for hardening containers
- Understand how misconfigurations can compromise container isolation
- Learn best practices for building container images
- Identify container images that have known software vulnerabilities
- Leverage secure connections between containers
- Use security tooling to prevent attacks on your deployment

"Liz's 'how it works' approach to topics like process isolation, image security, and key Linux concepts will be extremely valuable to the IT practitioner looking for quality information. I highly recommend getting a copy of this book and following the security principles and guidance Liz has expertly described!"

—**Phil Estes**
Distinguished Engineer & CTO, Linux and Container Strategy, IBM Cloud

"The definitive guide to Linux kernel, container, and VM isolation in Liz's inimitable style. If you enjoy her 'from scratch' talks and demos, this book will take you deeper into the mystical world of container security with illuminating tips, tools, and techniques to keep your applications and data safe."

—**Andrew Martin**
Director, ControlPlane

Liz Rice is VP Open Source Engineering with Aqua Security, chair of the CNCF's Technical Oversight Committee, and served as co-chair of the KubeCon + CloudNativeCon 2018 events in Copenhagen, Shanghai, and Seattle.

SECURITY / SYS ADMIN

US $49.99 CAN $65.99
ISBN: 978-1-492-05670-6

Twitter: @oreillymedia
facebook.com/oreilly

Preface

Many organizations are running applications in cloud native environments, using containers and orchestration to facilitate scalability and resilience. If you're a member of the Operations, the DevOps, or even the DevSecOps team setting up these environments for your company, how do you know whether your deployments are secure? If you're a security professional with experience in traditional server-based or virtual machine–based systems, how can you adapt your existing knowledge for container-based deployments? And as a developer in the cloud native world, what do you need to think about to improve the security of your containerized applications? This book delves into some of the key underlying technologies that containers and cloud native rely on, to leave you better equipped to assess the security risks and potential solutions applicable to your environment and to help you avoid falling into bad practices that will leave your technology deployments exposed.

In this book you will learn about many of the building block technologies and mechanisms that are commonly used in container-based systems, and how they are constructed in the Linux operating system. Together we will dive deep into the underpinnings of how containers work and how they communicate so that you are well versed not just in the "what" of container security but also, and more importantly, in the "why." My goal in writing this book is to help you better understand what's happening when you deploy containers. I want to encourage you to build mental models that allow you to make your own assessment of potential security risks that could affect your deployments.

This book primarily considers the kind of "application containers" that many businesses are using these days to run their business applications in systems such as Kubernetes and Docker. This is in contrast to "system containers" such as LXC and LXD from the Linux Containers Project (*https://linuxcontainers.org*). In an application container, you are encouraged to run immutable containers with as little code as is necessary to run the application, whereas in a system container environment the idea is to run an entire Linux distribution and treat it more like a virtual machine. It's considered perfectly normal to SSH into a system container, but application container

security experts will look at you askance if you want to SSH into an application container (for reasons covered later in this book). However, the basic mechanisms used to create application and system containers alike are control groups, namespaces, and changing the root directory, so this book will give you a solid foundation from which you may wish to explore the differences in approach taken by the different container projects.

Who This Book Is For

Whether you consider yourself a developer, a security professional, an operator, or a manager, this book will suit you best if you like to get into the nitty-gritty of how things work, and if you enjoy time spent in a Linux terminal.

If you are looking for an instruction manual that gives a step-by-step guide to securing containers, this may not be the book for you. I don't believe there is a one-size-fits-all approach that would work for every application in every environment and every organization. Instead, I want to help you understand what is happening when you run applications in containers, and how different security mechanisms work, so that you can judge the risks for yourself.

As you'll find out later in this book, containers are made with a combination of features from the Linux kernel. Securing containers involves using a lot of the same mechanisms as you would use on a Linux host. (I use the term "host" to cover both virtual machines and bare-metal servers.) I lay out how these mechanisms work and then show how they apply in containers. If you are an experienced system administrator, you'll be able to skip over some sections to get to the container-specific information.

I assume that you have some basic familiarity with containers, and you have probably at least toyed with Docker or Kubernetes. You will understand terms like "pulling a container image from a registry" or "running a container" even if you don't know exactly what is happening under the covers when you take these actions. I don't expect you to know the details of how containers work—at least, not until you have read the book.

What This Book Covers

We'll start in Chapter 1 by considering threat models and attack vectors that affect container deployments, and the aspects that differentiate container security from traditional deployment security. The remainder of the book is concerned with helping you build a thorough understanding of containers and these container-specific threats, and with how you can defend against them.

Before you can really think about how to secure containers, you'll need to know how they work. Chapter 2 sets the scene by describing some core Linux mechanisms such as system calls and capabilities that will come into play when we use containers. Then in Chapters 3 and 4, we'll delve into the Linux constructs that containers are made from. This will give you an understanding of what containers really are and of the extent to which they are isolated from each other. We'll compare this with virtual machine isolation in Chapter 5.

In Chapter 6 you'll learn about the contents of container images and consider some best practices for building them securely. Chapter 7 addresses the need to identify container images with known software vulnerabilities.

In Chapter 8 we will look at some optional Linux security measures that can be applied to harden containers beyond the basic implementation we saw in Chapter 4. We will look into ways that container isolation can be compromised through dangerous but commonplace misconfigurations in Chapter 9.

Then we will turn to the communications between containers. Chapter 10 looks at how containers communicate and explores ways to leverage the connections between them to improve security. Chapter 11 explains the basics of keys and certificates, which containerized components can use to identify each other and set up secure network connections between themselves. This is no different for containers than it is for any other component, but this topic is included since keys and certificates are often a source of confusion in distributed systems. In Chapter 12 we will see how certificates and other credentials can be safely (or not so safely) passed to containers at runtime.

In Chapter 13 we will consider ways in which security tooling can prevent attacks at runtime, taking advantage of the features of containers.

Finally, Chapter 14 reviews the top 10 security risks published by the Open Web Application Security Project and considers container-specific approaches for addressing them. Spoiler alert: some of the top security risks are addressed in exactly the same way whether your application is containerized or not.

A Note about Kubernetes

These days the majority of folks using containers are doing so under the Kubernetes (*https://kubernetes.io*) orchestrator. An orchestrator automates the process of running different workloads in a cluster of machines, and there are places in this book where I will assume that you have a basic grasp of this concept. In general, I have tried to stay focused on concepts that act at the level of the underlying containers—the "data plane" in a Kubernetes deployment.

Because Kubernetes workloads run in containers, this book is relevant to Kubernetes security, but it is not a comprehensive treatment of everything related to securing Kubernetes or cloud native deployments. There are many other concerns around the configuration and use of the control plane components that are outside the scope of this book. If you would like more on this topic, you might be interested in the O'Reilly *Kubernetes Security* report (*https://oreil.ly/Of6yK*) (which I coauthored with Michael Hausenblas).

Examples

There are lots of examples in this book, and I encourage you to try them out for yourself.

In the examples I assume that you are comfortable with basic Linux command-line tools like `ps` and `grep`, and with the basic day-to-day activities of running container applications through the use of tools like `kubectl` or `docker`. This book will use the former set of tools to explain a lot more about what's happening when you use the latter!

To follow along with the examples in this book, you will need access to a Linux machine or virtual machine. I created the examples using an Ubuntu 19.04 virtual machine running under VirtualBox (*https://www.virtualbox.org/*) on my Mac; I also use Vagrant (*https://www.vagrantup.com/*) to create, start, and stop my virtual machines. You should be able to achieve similar results on different Linux distributions and using virtual machines from your favorite cloud provider.

How to Run Containers

For many people, their main (perhaps only) experience of running containers directly is with Docker. Docker democratized the use of containers by providing a set of tools that developers generally found easy to use. From a terminal, you manipulate containers and container images using the `docker` command.

This `docker` tool is really a thin layer making API calls to Docker's main component: a daemon that does all the hard work. Within the daemon is a component called `containerd` that is invoked whenever you want to run a container. The `containerd` component makes sure that the container image you want to run is in place, and it then calls a `runc` component to do the business of actually instantiating a container.

If you want to, you can run a container yourself by invoking `containerd` or even `runc` directly. The `containerd` project was donated by Docker to the Cloud Native Computing Foundation (*https://cncf.io*) (CNCF) back in 2017.

Kubernetes uses an interface called the Container Runtime Interface (CRI) beneath which users can opt for a container runtime of their choice. The most commonly used options today are the aforementioned `containerd` (*https://containerd.io*) and `CRI-O` (*https://cri-o.io*) (which originated from Red Hat before being donated to the CNCF).

The `docker` CLI is just one option for managing containers and images. There are several others you can use to run the kind of application containers covered in this book. Red Hat's `podman` tool, originally conceived to remove reliance on a daemon component, is one such option.

The examples in this book use a variety of different container tools to illustrate that there are multiple container implementations that share many common features.

Feedback

There is a website at *containersecurity.tech* to accompany this book. You are invited to raise issues there with feedback and any corrections that you'd like to see in future editions.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a general note.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://containersecurity.tech*.

If you have a technical question or a problem using the code examples, please send email to *bookquestions@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Container Security* by Liz Rice (O'Reilly). Copyright 2020 Vertical Shift Ltd., 978-1-492-05670-6."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/container-security*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For more information about our books, courses, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

I'm grateful to many people who have helped and supported me through the process of writing this book.

- My editor at O'Reilly, Virginia Wilson, for keeping everything on track and making sure the book is up to scratch.
- The technical reviewers who provided thoughtful comments and actionable feedback: Akhil Behl, Alex Pollitt, Andrew Martin, Erik St. Martin, Phil Estes, Rani Osnat, and Robert P. J. Day.
- My colleagues at Aqua Security who taught me so much about container security over the years.
- Phil Pearl—my husband, my best critic and coach, and my best friend.

CHAPTER 1

Container Security Threats

In the last few years, the use of containers has exploded. The concepts around containers existed for several years before Docker, but most observers agree that it was Docker's easy-to-use command-line tools that started to popularize containers among the developer community from its launch in 2013.

Containers bring many advantages: as described in Docker's original tagline, they allow you to "build once, run anywhere." They do this by bundling together an application and all its dependencies and isolating the application from the rest of the machine it's running on. The containerized application has everything it needs, and it is easy to package up as a container image that will run the same on my laptop and yours, or in a server in a data center.

A knock-on effect of this isolation is that you can run multiple different containers side by side without them interfering with each other. Before containers, you could easily end up with a dependency nightmare where two applications required different versions of the same packages. The easiest solution to this problem was simply to run the applications on separate machines. With containers, the dependencies are isolated from each other so it becomes straightforward to run multiple apps on the same server. People quickly realized that they could take advantage of containerization to run multiple applications on the same host (whether it's a virtual machine or a bare-metal server) without having to worry about dependencies.

The next logical step was to spread containerized applications across a cluster of servers. Orchestrators such as Kubernetes automate this process so that you no longer have to manually install an app on a particular machine; you tell the orchestrator what containers you want to run, and it finds a suitable location for each one.

From a security perspective, many things are the same in a containerized environment as they are in a traditional deployment. There are attackers out in the world who want to steal data, or modify the way a system behaves, or use other people's compute resources to mine their own cryptocurrencies. This doesn't change when you move to containers. However, containers do change a lot about the way that applications run, and there are a different set of risks as a result.

Risks, Threats, and Mitigations

A *risk* is a potential problem, and the effects of that problem if it were to occur.

A *threat* is a path to that risk occurring.

A *mitigation* is a countermeasure against a threat—something you can do to prevent the threat or at least reduce the likelihood of its success.

For example, there is a risk that someone could steal your car keys from your house and thus drive off in your car. The threats would be the different ways they might steal the keys: breaking a window to reach in and pick them up; putting a fishing rod through your letter box; knocking on your door and distracting you while an accomplice slips in quickly to grab the keys. A mitigation for all these threats might be to keep your car keys out of sight.

Risks vary greatly from one organization to another. For a bank holding money on behalf of customers, the biggest risk is almost certainly that money being stolen. An ecommerce organization will worry about the risks of fraudulent transactions. An individual running a personal blog site might fear someone breaking in to impersonate them and post inappropriate comments. Because privacy regulations differ between nations, the risk of leaking customers' personal data varies with geography—in many countries the risk is "only" reputational, while in Europe the General Data Protection Regulation (GDPR) allows for fines of up to 4% of a company's total revenue (*https://oreil.ly/guQg3*).

Because the risks vary greatly, the relative importance of different threats will also vary, as will the appropriate set of mitigations. A risk management framework is a process for thinking about risks in a systematic way, enumerating the possible threats, prioritizing their importance, and defining an approach to mitigation.

Threat modeling is a process of identifying and enumerating the potential threats to a system. By systematically looking at the system's components and the possible modes of attack, a threat model can help you identify where your system is most vulnerable to attack.

There is no single comprehensive threat model, as it depends on your risks, your particular environment, your organization, and the applications you're running, but it is possible to list some potential threats that are common to most, if not all, container deployments.

Container Threat Model

One way to start thinking about the threat model is to consider the actors involved. These might include:

- *External attackers* attempting to access a deployment from outside
- *Internal attackers* who have managed to access some part of the deployment
- *Malicious internal actors* such as developers and administrators who have some level of privilege to access the deployment
- *Inadvertent internal actors* who may accidentally cause problems
- *Application processes* that, while not sentient beings intending to compromise your system, might have programmatic access to the system

Each actor has a certain set of permissions that you need to consider:

- What access do they have through credentials? For example, do they have access to user accounts on the host machines your deployment is running on?
- What permissions do they have on the system? In Kubernetes, this could refer to the role-based access control settings for each user, as well as anonymous users.
- What network access do they have? For example, which parts of the system are included within a Virtual Private Cloud (VPC)?

There are several possible routes to attacking a containerized deployment, and one way to map them is to think of the potential attack vectors at each stage of a container's life cycle. These are summarized in Figure 1-1.

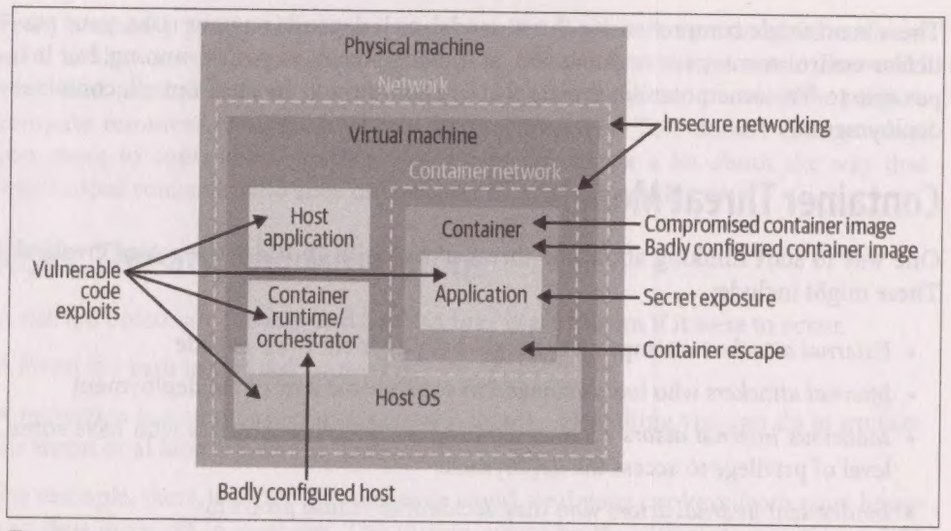


Figure 1-1. Container attack vectors

Vulnerable application code
: The life cycle starts with the application code that a developer writes. This code, and the third-party dependencies that it relies on, can include flaws known as vulnerabilities, and there are thousands of published vulnerabilities that an attacker can exploit if they are present in an application. The best way to avoid running containers with known vulnerabilities is to scan images, as you will see in Chapter 7. This isn't a one-off activity, because new vulnerabilities are discovered in existing code all the time. The scanning process also needs to identify when containers are running with out-of-date packages that need to be updated for security patches. Some scanners can also identify malware that has been built into an image.

Badly configured container images
: Once the code has been written, it gets built into a container image. When you are configuring how a container image is going to be built, there are plenty of opportunities to introduce weaknesses that can later be used to attack the running container. These include configuring the container to run as the root user, giving it more privilege on the host than it really needs. You'll read more about this in Chapter 6.

Build machine attacks
: If an attacker can modify or influence the way a container image is built, they could insert malicious code that will subsequently get run in the production environment. In addition, finding a foothold within the build environment could be a stepping stone toward breaching the production environment. This is also discussed in Chapter 6.

Supply chain attacks
: Once the container image is built, it gets stored in a registry, and it gets retrieved or "pulled" from the registry at the point where it's going to be run. How do you know that the image you pull is exactly the same as what you pushed earlier? Could it have been tampered with? An actor who can replace an image or modify an image between build and deployment has the ability to run arbitrary code on your deployment. This is another topic I'll cover in Chapter 6.

Badly configured containers
: As we'll discuss in Chapter 9, it's possible to run containers with settings that give it unnecessary, and perhaps unplanned, privileges. If you download YAML configuration files from the internet, please don't run them without carefully checking that they do not include insecure settings!

Vulnerable hosts
: Containers run on host machines, and you need to ensure that those hosts are not running vulnerable code (for example, old versions of orchestration components with known vulnerabilities). It's a good idea to minimize the amount of software installed on each host to reduce the attack surface, and hosts also need to be configured correctly according to security best practices. This is discussed in Chapter 4.

Exposed secrets
: Application code often needs credentials, tokens, or passwords in order to communicate with other components in a system. In a containerized deployment, you need to be able to pass these secret values into the containerized code. As you'll see in Chapter 12, there are different approaches to this, with varying levels of security.

Insecure networking
: Containers generally need to communicate with other containers or with the outside world. Chapter 10 discusses how networking works in containers, and Chapter 11 discusses setting up secure connections between components.

Container escape vulnerabilities
: The widely used container runtimes including `containerd` and `CRI-O` are by now pretty battle-hardened, but it's still within the realm of possibility that there are bugs yet to be found that would let malicious code running inside a container escape out onto the host. One such issue, sometimes referred to as Runcescape (*https://oreil.ly/cFSaJ*), came to light as recently as 2019. You'll read about the isolation that is supposed to keep application code constrained within a container in Chapter 4. For some applications, the consequences of an escape could be

sufficiently damaging that it's worth considering stronger isolation mechanisms, such as those covered in Chapter 8.

There are also some attack vectors that are outside the scope of this book.

- Source code is generally held in repositories, which could conceivably be attacked in order to poison the application. You will need to ensure that user access to the repository is controlled appropriately.
- Hosts are networked together, often using a VPC for security, and typically connected to the internet. Exactly as in a traditional deployment, you need to protect the host machines (or virtual machines) from access by threat actors. Secure network configuration, firewalling, and identity and access management all still apply in a cloud native deployment as they do in a traditional deployment.
- Containers typically run under an orchestrator—commonly Kubernetes in today's deployments, though there are other options such as Docker Swarm or Hashicorp Nomad. If the orchestrator is configured insecurely or if administrative access is not controlled effectively, this gives attackers additional vectors to affect the deployment.

For more on threat models in Kubernetes deployments, you may be interested in reading the Kubernetes Threat Model (*https://oreil.ly/r0ZAG*) commissioned by the CNCF.

In addition, the CNCF's Financial User Group has published a Kubernetes Attack Tree (*https://oreil.ly/r1lg8*) created using the STRIDE (*https://oreil.ly/rNmPN*) methodology.

Security Boundaries

A security boundary (sometimes called a trust boundary) appears between parts of the system, such that you would need some different set of permissions to move between those parts. Sometimes these boundaries are set up administratively—for example, in a Linux system, the system administrator can modify the security boundary defining what files a user can access by changing the groups that the user is a member of. If you are rusty on Linux file permissions, a refresher is coming up in Chapter 2.

A container is a security boundary. Application code is supposed to run within that container, and it should not be able to access code or data outside of the container except where it has explicitly been given permission to do so (for example, through a volume mounted into the container).

The more security boundaries there are between an attacker and their target (your customer data, for example), the harder it is for them to reach that target.

The attack vectors described in "Container Threat Model" on page 3 can be chained together to breach several security boundaries. For example:

- An attacker may find that because of a vulnerability in an application dependency, they are able to execute code remotely within a container.
- Suppose that the breached container doesn't have direct access to any data of value. The attacker needs to find a way to move out of the container, either to another container or to the host. A container escape vulnerability would be one route out of the container; insecure configuration of that container could provide another. If the attacker finds either of these routes available, they can now access the host.
- The next step would be to look for ways to gain root privileges on the host. This step might be trivial if your application code is running as root inside the container, as you'll see in Chapter 4.
- With root privileges on the host machine, the attacker can get to anything that the host, or any of the containers running on that host, can reach.

Adding and strengthening the security boundaries in your deployment will make life more difficult for the attacker.

An important aspect of the threat model is to consider the possibility of attacks from within the environment in which your applications are running. In cloud deployments, you may be sharing some resources with other users and their applications. Sharing machine resources is called *multitenancy*, and it has a significant bearing on the threat model.

Multitenancy

In a multitenant environment, different users, or *tenants*, run their workloads on shared hardware. (You may also come across the term "multitenancy" in a software application context, where it refers to multiple users sharing the same instance of software, but for the purposes of this discussion, only the hardware is shared.) Depending on who owns those different workloads and how much the different tenants trust each other, you might need stronger boundaries between them to prevent them from interfering with each other.

Multitenancy is a concept that has been around since the mainframe days in the 1960s, when customers rented CPU time, memory, and storage on a shared machine. This is not so very different from today's public clouds, like Amazon AWS, Microsoft Azure, and Google Cloud Platform, where customers rent CPU time, memory, and

storage, along with other features and managed services. Since Amazon AWS launched EC2 back in 2006, we have been able to rent virtual machine instances running on racks of servers in data centers around the world. There may be many virtual machines (VMs) running on a physical machine, and as a cloud customer operating a set of VMs you have no idea who is operating the VMs that neighbor yours.

Shared Machines

There are situations in which a single Linux machine (or virtual machine) may be shared by many users. This is very common in university settings, for instance, and this is a good example of true multitenancy, where users don't trust each other and, quite frankly, the system administrators don't trust the users. In this environment Linux access controls are used to strictly limit user access. Each user has their own login ID, and the access controls of Linux are used to limit access to ensure, for example, that users can modify only files in their own directories. Can you imagine the chaos if university students could read, or—even worse—modify, their classmates' files?

As you'll see in Chapter 4, all the containers running on the same host share the same kernel. If the machine is running the Docker daemon, any user who can issue `docker` commands effectively has root access, so a system administrator won't want to grant that to untrusted users.

In enterprise situations, and more specifically in cloud native environments, you are less likely to see this kind of shared machine. Instead, users (or teams of users who trust each other) will typically use their own resources allocated to them in the form of virtual machines.

Virtualization

Generally speaking, virtual machines are considered to be pretty strongly isolated from each other, by which we mean that it's unlikely that your neighbors can observe or interfere with the activities in your VMs. You can read more about how this isolation is achieved in Chapter 5. In fact, according to the accepted definition (*https://oreil.ly/yfkQI*), virtualization doesn't count as multitenancy at all: multitenancy is when different groups of people share a single instance of the same software, and in virtualization the users don't have access to the hypervisor that manages their virtual machines, so they don't share any software.

That's not to say that the isolation between virtual machines is perfect, and historically users have complained about "noisy neighbor" issues, where the fact that you are sharing a physical machine with other users can result in unexpected variances in performance. Netflix was an early adopter of the public cloud, and in the section "Cotenancy is hard" (*https://oreil.ly/CGlZ0*) in a 2010 blog post, it acknowledged that it built systems that might deliberately abandon a subtask if it proved to be operating too slowly. More recently, others have claimed that the noisy neighbor problem isn't a real issue (*https://oreil.ly/iE4qE*).

There have also been cases of software vulnerabilities that could compromise the boundary between virtual machines.

For some applications and some organizations (especially government, financial, or healthcare), the consequences of a security breach are sufficiently serious to warrant full physical separation. You can operate a private cloud, running in your own data center or managed by a service provider on your behalf, to ensure total isolation of your workloads. Private clouds sometimes come with additional security features such as additional background checks on the personnel who have access to the data center.

Many cloud providers have VM options where you are guaranteed to be the only customer on a physical machine. It's also possible to rent bare-metal machines operated by cloud providers. In both these scenarios, you will completely avoid the noisy neighbor issue, and you also have the advantage of the stronger security isolation between physical machines.

Whether you are renting physical or virtual machines in the cloud or using your own servers, if you're running containers, you may need to consider the security boundaries between multiple groups of users.

Container Multitenancy

As you'll see in Chapter 4, the isolation between containers is not as strong as that between VMs. While it does depend on your risk profile, it's unlikely that you want to use containers on the same machine as a party that you don't trust.

Even if all the containers running on your machines are run by you or by people you absolutely trust, you might still want to mitigate against the fallibility of humans by making sure that your containers can't interfere with each other.

In Kubernetes, you can use *namespaces* to subdivide a cluster of machines for use by different individuals, teams, or applications.

The word "namespace" is an overloaded term. In Kubernetes, a namespace is a high-level abstraction that subdivides cluster resources that can have different Kubernetes access controls applied to them. In Linux, a namespace is a low-level mechanism for isolating the machine resources that a process is aware of. You'll learn about this kind of namespace in detail in Chapter 4.

Use role-based access control (RBAC) to limit the people and components that can access these different Kubernetes namespaces. The details of how to do this are

outside the scope of this book, but I would like to mention that Kubernetes RBAC controls only the actions you can perform through the Kubernetes API. Application containers in Kubernetes pods that happen to be running on the same host are protected from each other only by container isolation, as described in this book, even if they are in different namespaces. If an attacker can escape a container to the host, the Kubernetes namespace boundary makes not one jot of difference to their ability to affect other containers.

Container Instances

Cloud services such as Amazon AWS, Microsoft Azure, or Google Cloud Platform offer many *managed services*, through which the user can rent software, storage, and other components without having to install or manage them. A classic example is Amazon's Relational Database Service (RDS); with RDS, you can easily provision databases that use well-known software like PostgreSQL, and getting your data backed up is as simple as ticking a box (and paying a bill, of course).

Managed services have extended to the world of containers, too. Azure Container Instances and AWS Fargate are services that allow you to run containers without having to worry about the underlying machine (or virtual machine) on which they run.

This can save you from a significant management burden and allows you to easily scale the deployment at will. However, at least in theory, your container instances could be colocated on the same virtual machine as those of other customers. Check with your cloud provider if in doubt.

You are now aware of a good number of potential threats to your deployment. Before we dive into the rest of the book, I'd like to introduce some basic security principles that should guide your thinking when assessing what security tools and processes you need to introduce into your deployment.

Security Principles

These are general guidelines that are commonly considered to be a wise approach regardless of the details of what you're trying to secure.

Least Privilege

The principle of least privilege states that you should limit access to the bare minimum that a person or component needs in order to do their job. For example, if you have a microservice that performs product search in an ecommerce application, the principle of least privilege suggests that the microservice should only have credentials that give it read-only access to the product database. It has no need to access, say, user or payment information, and it has no need to write product information.

Defense in Depth

As you'll see in this book, there are many different ways you can improve the security of your deployment and the applications running within it. The principle of defense in depth tells us that you should apply layers of protection. If an attacker is able to breach one defense, another layer should prevent them from harming your deployment or exfiltrating your data.

Reducing the Attack Surface

As a general rule, the more complex a system is, the more likely it is that there is a way to attack it. Eliminating complexity can make the system harder to attack. This includes:

- Reducing access points by keeping interfaces small and simple where possible
- Limiting the users and components who can access a service
- Minimizing the amount of code

Limiting the Blast Radius

The concept of segmenting security controls into smaller subcomponents or "cells" means that should the worst happen, the impact is limited. Containers are well-suited to this principle, because by dividing an architecture into many instances of a microservice, the container itself can act as a security boundary.

Segregation of Duties

Related to both least privilege and limiting blast radius is the idea of segregating duties so that, as much as possible, different components or people are given authority over only the smallest subset of the overall system that they need. This approach limits the damage a single privileged user might inflict by ensuring that certain operations require more than one user's authority.

Applying Security Principles with Containers

As you'll see in later sections of this book, the granularity of containers can help us in the application of all these security principles.

Least privilege
: You can give different containers different sets of privileges, each minimized to the smallest set of permissions it needs to fulfill its function.

Defense in depth
: Containers give another boundary where security protections can be enforced.